TRABAJO FIN DE GRADO

INTEGRACIÓN SENSORES DE PROXIMIDAD EN ROBOT ASISTENCIAL AMOR

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Elena Jiménez Martín

Tutor: Edwin Daniel Oña Simbaña

Leganés, 26 de septiembre de 2017

Título: INTEGRACIÓN SENSORES DE PROXIMIDAD EN ROBOT ASISTENCIAL AMOR

Autor: Elena Jiménez Martín

Tutor: Edwin Daniel Oña Simbaña

EL TRIBUNAL

Presidente: Pedro Martín Mateos

Vocal: Juan Miguel García Haro

Secretario: Ignacio Rubio Díaz

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 10 de Octubre de 2017 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mi tutor Edwin el darme la oportunidad de realizar este trabajo, así como todo el apoyo brindado durante su desarrollo. Del mismo modo, agradecer a Bartek y a Jorge su ayuda incondicional durante todo este tiempo, ya que sin ellos este proyecto no hubiera sido posible.

Gracias a mis amigos y compañeros de universidad, que me han acompañado a lo largo de este camino, disfrutando de los buenos momentos como levantándome en los malos, haciendo de estos mis mejores años.

Agradecer a mis compañeros de trabajo su apoyo y paciencia durante estos meses de esfuerzo, haciendo mi camino mucho más fácil.

Por último, agradecer a mi familia, y en especial a mi madre, todo lo que ha hecho por mí durante este tiempo para que esto fuera posible. Aunque no ha sido fácil, sé que ha merecido la pena. A mi padre, que sé que estaría orgulloso de lo que he conseguido, esto es para ti.

RESUMEN

La Universidad Carlos III de Madrid adquirió en 2013 el robot asistencial AMOR, cuya finalidad es asistir en tareas cotidianas o de rehabilitación a personas de la tercera edad o con algún tipo de discapacidad.

El objetivo de este proyecto es desarrollar una aplicación de prevención de colisiones, que garantice la seguridad durante el manejo del robot por el usuario.

Para este fin, se ha desarrollado un programa ejecutable en el entorno Linux, que recibe la respuesta de unos sensores de proximidad integrados en el robot. Dichos sensores están distribuidos a lo largo del robot para sensar el espacio cercano alrededor del mismo. A partir de los datos de los sensores, el programa establece unos niveles de alerta en función de la distancia de los obstáculos del entorno, y configura las acciones necesarias en los actuadores del robot para evitar que colisione.

Finalmente, este módulo podrá integrarse como parte de otras aplicaciones para la manipulación de AMOR u otros robots, permitiendo que todas ellas cuenten con un sistema de seguridad ante obstáculos.

Palabras clave: Robot, asistencial, AMOR, C++, YARP, Arduino, sensores, proximidad, obstáculos, colisión.

ABSTRACT

The Carlos III University of Madrid acquired the assistive robot AMOR in 2013; its purpose is to assist in daily tasks or rehabilitation of elderly people or those with some type of disability.

The objective of this project is to develop a collision prevention application, which guarantees safety during the robot's handling by the user.

To this end, an executable program has been developed in a Linux environment, which receives the input of several proximity sensors integrated in the robot. These sensors are distributed along the robot to sense the near space around it. From the data of the sensors, the program sets different alert levels based on the distance of the obstacles in the environment, and generates the necessary actions on the robot's actuators to avoid collision.

Finally, this module can be integrated as part of other applications for the manipulation of AMOR or other robots, allowing all of them to include an obstacle detection safety system.

Keywords: Robot, assistive, care, AMOR, C++, YARP, Arduino, sensors, proximity, obstacles, collision.

Índice

Índice de figuras

1. Introducción

Los últimos análisis realizados a nivel mundial en 188 países muestran que la esperanza de vida para ambos sexos ha aumentado considerablemente, lo que da lugar a tratar de buscar soluciones para dotar de mejoras en la calidad de vida a una numerosa población de avanzada edad, o personas que presenten algún tipo de discapacidad. [1]

Debido a esto, el número de investigaciones para el desarrollo tecnológico ha ido en aumento, encontrándose entre éstas el campo de la robótica asistencial, dirigido a ayudar a las personas en sus tareas cotidianas o en tratamientos de rehabilitación.

Un factor muy importante a tener en cuenta a la hora de introducir estos robots en el día a día de las personas, es la seguridad. Es necesario hacer una evaluación de los posibles riesgos que pueden existir, para no entorpecer o dañar al propio usuario.

Este proyecto propone una solución al problema de la aparición de obstáculos en la trayectoria del robot, realizando pruebas en un entorno real, para evitar daños en el ambiente o sobre el propio individuo, logrando que el robot disminuya su velocidad o se detenga frente un obstáculo. Finalmente, se implementará una interfaz gráfica que nos permita comprobar la fiabilidad del funcionamiento del sistema, así como se proporcionaran durante la memoria los resultados obtenidos en las diferentes pruebas.

2. Objetivo

El objetivo de este proyecto es diseñar un plug in o ejecutable que evite colisiones frente a obstáculos, para posteriormente implementarla como un módulo dentro de una aplicación principal para el control del robot AMOR. De esta manera, se plantea una solución ante el peligro existente de que cause daños durante el desarrollo de sus tareas asistenciales, tanto en el ambiente en el que esté prestando servicio, como sobre el propio usuario, mediante la detección de obstáculos en su trayectoria con la ayuda de sensores. Cuando el sistema detecte que una persona u objeto le está cortando el paso, este reducirá su velocidad o realizará una parada de emergencia, en función de la distancia a la que se encuentre el mismo.

Para llevar a cabo este objetivo, el robot cuenta con una red de sensores infrarrojos de proximidad, cuyo módulo de adquisición fue implementado durante el desarrollo de un proyecto previo. A lo largo de este trabajo se procesarán las señales captadas por estos sensores, caracterizándolas para nuestro objetivo específico, y se realizarán diversas pruebas para analizar los rangos de medida, fiabilidad, diferenciación de la respuesta según el material del obstáculo encontrado, entre otras.

Por último, se programará al robot para que actúe de una manera concreta en función del obstáculo que encuentre, especificando distintos niveles de alerta según nuestras necesidades.

Para el desarrollo de este proyecto se ha empleado el equipo disponible en el Laboratorio de Robótica Asistencial de la Universidad Carlos III de Madrid (Figura 1), que cuenta con el robot y sus accesorios en las instalaciones, así como los programas necesarios para su configuración, además de tener la representación de un entorno real de una vivienda, contando con una cocina para la realización de las pruebas.

Figura 1. Laboratorio de Robótica Asistencial (UC3M)

Durante la descripción de la memoria se irán especificando los detalles para cada una de las pruebas y resultados obtenidos. El código desarrollado está disponible en los anexos o a través de enlaces a la plataforma Github [25][26][27].

3. Estado del arte

En la actualidad la robótica asistencial es un área de investigación donde la interacción humano - robot es determinante. Esta área implica la creación de algoritmos de visión e inteligencia artificial, técnicas de control, e incluso, el estudio de factores psicológicos para medir el impacto del robot en la tarea de interacción. [2]

En cuanto a robots asistenciales, existen algunos modelos que se han desarrollado. Entre estos modelos se tiene a Novel Assitive Robot, un robot capaz de asistir al usuario en las tareas de auto alimentación (ver Figura 2A). Otro robot destinado a asistencia de pacientes con problemas de movilidad y destinado a la auto alimentación y a asistir en tareas como el cuidado y el aseo personal es ASIBOT (ver Figura 2B).

Figura 2. Robots asistenciales. (A) Novel Assistive Robot (B) ASIBOT

ASIBOT es un robot asistencial portátil destinado a personas de la tercera edad o con algún tipo de discapacidad, que permite la realización de tareas cotidianas como comer, beber, afeitarse, etc. El robot está dotado de 5 grados de libertad, 10 kilos de peso, 1,3 m de alcance y puede levantar cargas de hasta 2 kg. ASIBOT se encuentra bajo experimentación en el Hospital Nacional de Parapléjicos de Toledo, y en la Universidad Carlos III tiene lugar la investigación y desarrollo de nuevas aplicaciones para mejorar la funcionalidad del mismo.

En el año 2013, la Universidad Carlos III de Madrid adquirió el robot asistencial AMOR a la empresa EXACT DYNAMICS BV, robot en el que se basa el presente proyecto, y cuya finalidad es asistir en tareas cotidianas o de rehabilitación a personas mayores o con alguna discapacidad. Ese robot cuenta con 7 grados de libertad, dos más en comparación con ASIBOT, y resulta algo más ligero, con 9 kg de peso.

Durante estos años, se han desarrollado numerosas aplicaciones para el manejo de AMOR, mediante programas para su control por joystick, o mediante la reproducción de movimientos humanos a través de una cámara Kinect. Sin embargo, no se ha desarrollado todavía ninguna aplicación que abordase el tema de la seguridad frente a posibles colisiones con obstáculos. Únicamente se realizó un estudio mediante simulaciones para la implantación de sensores sobre ASIBOT, con el fin de desarrollar un sistema anti-colisiones, que no se ha llevado a cabo todavía. [3] En la Figura 3 podemos ver las simulaciones con la disposición de los sensores sobre ASIBOT.

Figura 3. Simulaciones para implementación de sensores en ASIBOT

Esta carencia en sistemas de seguridad sobre los robots asistenciales que se encuentran en los laboratorios de la universidad, es lo que ha motivado al planteamiento y realización de este proyecto.

A continuación, se expondrán los diferentes métodos de implementación de sistemas de seguridad que podemos encontrar en robots en la actualidad:

A) Diseño mecánico del robot.

- Seguridad intrínseca: El primer criterio importante para limitar los posibles daños por colisiones es reducir el peso de las partes móviles del robot. Un ejemplo de prototipo de este tipo de seguridad es el diseño del DLR-III Lightweight Robot [18], que es capaz de operar con una carga igual que su propio peso (13,5 kg).

Figura 4. DLR-III Lightweight Robot

Para la implementación de estos sistemas de seguridad se emplean materiales ligeros pero rígidos para facilitar el movimiento. En el DLR-III, la transmisión / reducción del motor se basa en armónicos, que muestran una alta relación de reducción y capacidad de transmisión de energía eficiente. Además, es recomendable ubicar todos los pesos relevantes, sobre todo los motores, en la base del robot.

- Accionamiento de impedancia variable o rigidez variable: las transmisiones totalmente acopladas pueden garantizar una interacción segura, pero resultan ineficientes en la transferencia de energía desde los actuadores a las articulaciones al tratar de reproducir un movimiento rápido. Un modo de obtener un alto rendimiento para garantizar la seguridad durante el movimiento en las articulaciones, es permitir que la transmisión sea variable durante la ejecución de tarea [24].

  Este diseño, conocido como "método de impedancia variable" o VIA por sus siglas en inglés, Variable Impedance Approach, es un diseño mecánico de control que permite variar rápidamente y de manera continua durante la ejecución de una tarea el valor de componentes tales como rigidez, amortiguación y relación de engranajes, garantizando un bajo nivel de riesgo de lesiones y minimizando efectos negativos en el rendimiento del control.

- Accionamiento distribuido: otro modo de reducir la inercia del brazo en los robots manipuladores para garantizar la seguridad preservando el rendimiento, es mediante la actuación macro-mini distribuida (DM2). Para cada grado de libertad (articulación), se emplean un par de actuadores conectados en paralelo y situados en diferentes partes del manipulador. La primera parte del DM2 es dividir la generación del par en actuadores de alta y baja frecuencia cuyo par sume en paralelo [22].

  La gravedad y otras fuerzas de poca variación en el tiempo se originan por actuadores pesados y de baja frecuencia, que se sitúan en la base del robot.

Para el accionamiento de los pares de alta frecuencia, se colocan pequeños motores en las articulaciones, garantizando un alto nivel de movimiento sin incrementar significativamente la impedancia combinada del sistema. Finalmente, logramos obtener una baja impedancia empleando un actuador elástico (SEA, Series Elastic Actuator).

B) Arquitectura de control

Otro modo de garantizar la seguridad frente a colisiones durante la manipulación del robot, es mediante la programación de un software de control. El software de control del robot suele estar estructurado de manera modular y jerárquica, lo que resulta ventajoso, además, para probar sus componentes de manera individual y aislar posibles fallos para lograr una robustez operativa (disponibilidad, fiabilidad y mantenibilidad).

Debido a la necesidad de monitoreo continuo del ambiente y del funcionamiento del robot, así como de cambios en la planificación de trayectorias, es necesario que el sistema operativo de una arquitectura de control se ejecute en tiempo real. Además, hay que tener en cuenta las restricciones de tiempo de ejecución, multitarea, priorización de procesos y niveles de interrupción.

C) Sensores

La integración de sensores en un robot manipulador está fuertemente ligado al punto anterior, ya que normalmente se integran como parte de un módulo del software de control. El uso de sensores (de proximidad, de temperatura, de fuerza, cámaras, etc) resulta muy útil, sobre todo, en robots móviles cuya base no se encuentre anclada a un punto.

Este método de seguridad frente a obstáculos resulta de gran ventaja, ya que permite que la operación del robot no esté limitada a un área específica, al ser capaz de detectar obstáculos independientemente del entorno en el que se encuentre el sistema.

D) Planificación de trayectorias en tiempo real

Consiste en un proceso off-line que determina un camino alternativo en tiempo real, replanificando la trayectoria del robot, si fuera posible, evitando el obstáculo detectado. Para ello, es necesario tener un conocimiento completo de la geometría estática del entorno. Para robots con muchos grados de libertad, el cálculo de una nueva trayectoria es muy complejo y requiere de mucho tiempo. Actualmente se está recurriendo a técnicas de probabilidad y estadística para poder abordar este problema.

La implementación del sistema de seguridad sobre AMOR que se desarrolla a lo largo de este proyecto, se basa en los sistemas de seguridad B) y C), al realizar la integración de sensores de proximidad dispuestos por todo el brazo robótico en un módulo de seguridad de una aplicación de control del robot.

Para finalizar, se planteará la implementación de un sistema de planificación de trayectorias (método D), como un posible proyecto futuro.

4. Elementos del proyecto

4.1. Hardware

A continuación, se detallarán los distintos componentes físicos que han formado parte del proyecto.

4.1.1. Robot AMOR (Universidad Carlos III de Madrid)

AMOR es un robot manipulador de servicio asistencial programable y motorizado [4]. Las distintas partes y enlaces que componen al robot coinciden con las de un brazo humano (hombro, codo y muñeca). Para simular el movimiento de un brazo humano real, se ha dotado al robot de siete grados de libertad.

Podemos distinguir seis enlaces principales controlados por siete actuadores, a saber:

- Columna
- Hombro
- Parte superior del brazo
- Codo
- Parte inferior del brazo
- Muñeca

En los enlaces situados entre hombro-codo y codo-muñeca, encontramos dos rotaciones adicionales. Todas estas uniones son accionadas eléctricamente por motores colocados lo más cerca posible de las mismas. En la Figura 5 podemos observar la numeración de las articulaciones y de los movimientos que puede efectuar el robot.

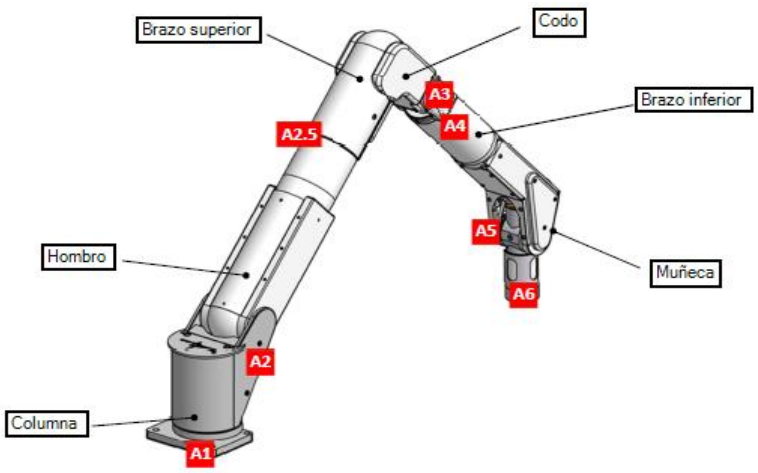

Figura 5. Disposición mecánica de AMOR

Características

- 7 grados de libertad
- Puede levantar un objeto de hasta 2,5 kg
- Diseñado para la seguridad. Incluye varias medidas de seguridad tales como un embrague mecánico deslizante, frenos, y funcionamiento a baja tensión
- Sin cableado externo
- Efector final modular con cableado de entrada /salida de uso general para conectar, por ejemplo, una cámara o un sensor de fuerza con retroalimentación
- Rotación de articulaciones ilimitada para tres de las siete uniones
- Alcance de hasta 85 cm.
- Funcionamiento a 24V (DC)

Requerimientos del sistema

Para el desarrollo de actividades con el robot se asume que el control se realiza con un sistema con las siguientes características:

- Microsoft Windows 2000 o superior
- Linux (Kernell ≥ 2.6.24)

En caso de utilizar otro tipo de plataformas, como por ejemplo Apple OS X o versiones anteriores a Linux Kernell, se deberá contactar con el distribuidor local para obtener información acerca de la comunicación con el sistema.

Complementos del robot

- Cable de alimentación con fusible (4A)

Figura 6. Cable de alimentación AMOR

- CAN cable de comunicación (D-SUB 9 macho-hembra, longitud 2,0 m)

Figura 7. Cable CAN de comunicación

- Adaptador de terminación CAN

Figura 8. Adaptador para cable CAN

- Pinza modular

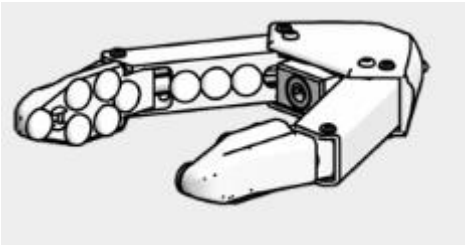

Figura 9. Pinza modular compatible con AMOR

4.1.2. Sensores infrarrojos Si1143

Estos sensores de proximidad infrarrojos evalúan la distancia de un objeto mediante la interpretación de la cantidad de luz reflejada por dicho objeto. Constan de unos diodos emisores infrarrojos, que emiten unos pulsos de luz de alta intensidad, y que, cuando un objeto los atraviesa, dichos pulsos son reflejados y captados por un fototransistor, que a su vez los enviará a un amplificador de corriente para interpretar la señal.

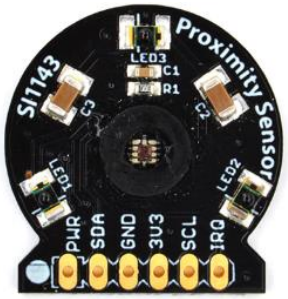

Figura 10. Sensor infrarrojo SI1143

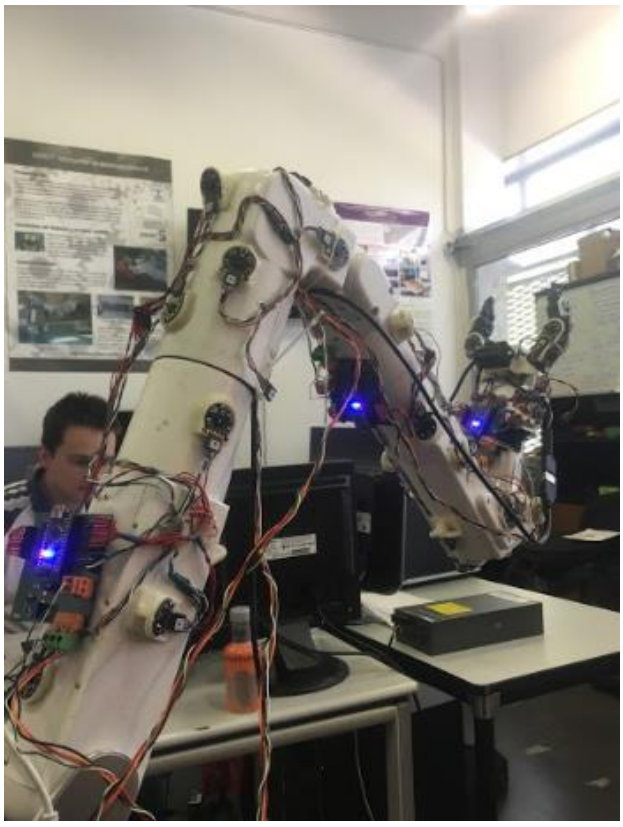

Figura 11. Disposición de los sensores en el robot

Descripción

El sensor SI1143 [5] es un sensor de proximidad y de luz ambiente de baja potencia basado en la reflectancia de la luz en los objetos. Presenta una interfaz digital de comunicación i²C, permite la programación de interrupciones, cuenta con un conversor analógico-digital y permite las mediciones en todo tipo de condiciones lumínicas. Este sensor mide la distancia a objetos situados un máximo entre 15 y 50 cm de distancia, dependiendo de las condiciones de luz del ambiente, y un

ángulo de amplitud de detección altamente efectivo entre los 20º y 45º, dependiendo de la posición de los leds del sensor.

Su tamaño, efectividad y posibilidades de configuración hacen que sea una perfecta elección para la medición de distancia según nuestra aplicación, ya que podemos configurarlo para adaptarlo a diferentes condiciones de luz dependiendo de donde se encuentre nuestro robot.

Para el desarrollo de este proyecto, se han implementado 16 sensores por cada zona del robot, dando un total de 48 sensores distribuidos por todo el brazo, para abarcar la mayor superficie de detección posible.

Características

El sensor dispone de 6 pines, de los cuales emplearemos para nuestro proyecto los siguientes:

- 3V3: Pin que proporciona la tensión requerida para la alimentación del circuito receptor. Se conectará al pin 3V3 de la placa Arduino.

- GND: Conexión al pin correspondiente a la tierra en la placa Arduino.

- SDA: I/O (bidireccional) que permite el intercambio de datos en la conexión maestro-esclavo i2C.

- SCL: Entrada que recibe la señal del reloj que permite la conexión i2C

La siguiente tabla recoge las especificaciones eléctricas recomendadas por el fabricante para el óptimo funcionamiento del sensor:

Parameter	Symbol	Test Condition	Min	Typ	Max	Unit
V_{DD} Supply Voltage	V_{DD}		1.71	—	3.6	V
V_{DD} OFF Supply Voltage	V_{DD_OFF}	OFF mode	–0.3		1.0	V
V_{DD} Supply Ripple Voltage		V_{DD} = 3.3 V 1 kHz–10 MHz	—	—	50	mVpp
Operating Temperature	T		–40	25	85	°C
SCL, SDA, Input High Logic Voltage	I^2C_{VIH}		V_{DD}x0.7	—	V_{DD}	V
SCL, SDA Input Low Logic Voltage	I^2C_{VIL}		0	—	V_{DD}x0.3	V
PS Operation under Direct Sunlight	Edc		—	—	128	klx
IrLED Emission Wavelength	I		750	850	950	nm
IrLED Supply Voltage	VLED	IrLED V_F = 1.0 V nominal	V_{DD}	—	4.3	V
IrLED Supply Ripple Voltage		Applies if IrLEDs use separate supply rail 0–30 kHz 30 kHz–100 MHz	 — —	 — —	 250 100	 mVpp mVpp
Start-Up Time		V_{DD} above 1.71 V	25	—	—	ms
LED3 Voltage		Start-up	V_{DD}x0.77	—	—	V

Figura 12. Especificaciones eléctricas sensores SI1143

En las siguientes imágenes se muestran un gráfico con las respuestas del sensor bajo diferentes condiciones ambientales, que servirán de apoyo a la hora de comparar los resultados de este proyecto, y un esquema del rango de detección de objetos dependiendo del ángulo en que se encuentren respecto al sensor.

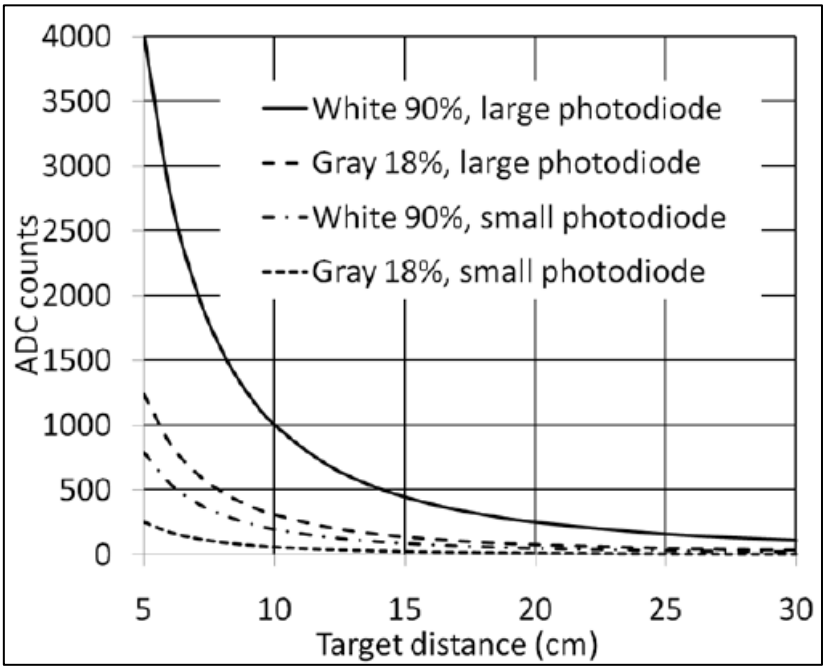

Figura 13. Respuestas del sensor SI1143

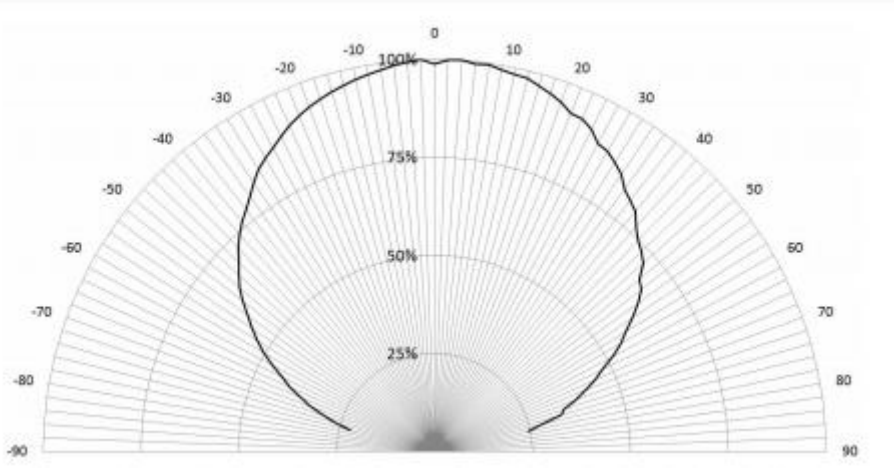

Figura 14. Respuesta del sensor SI1143 según la posición angular

4.1.3. Placas Arduino

Para la adquisición de los datos de los sensores se han empleado cuatro placas Arduino, distribuidas en el robot de la siguiente manera:

- Tres placas Arduino UNO, una para cada zona del robot (brazo, antebrazo y mano). Ha sido necesario colocar una placa por cada zona del robot, ya que de otro modo no disponíamos de pines suficientes en una sola placa para realizar el conexionado de todos los sensores.

- Una placa Arduino MEGA ADK, que actuará de placa maestra, agrupando los datos recopilados por las tres placas Arduino UNO. Esta será la encargada de realizar la transmisión de los datos al PC.

Arduino MEGA ADK

Se trata de una placa basada en el microcontrolador ATmega2560 con 54 pines I/O digitales, donde 15 de ellos pueden usarse como salidas PWM, 16 entradas analógicas, 4 UARTS, que son hardware de puerto serie, un oscilador a 16MHz, el conector de corriente, una conexión USB, header ICSP y un botón de reset.

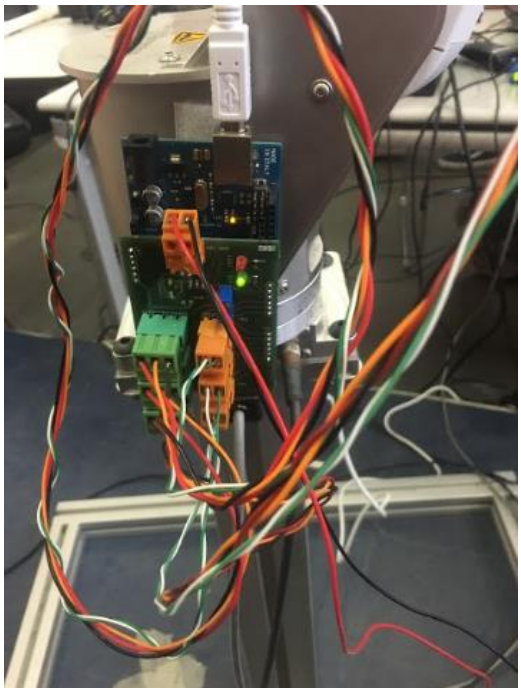

Figura 15. Placa Arduino MEGA ADK implementada sobre AMOR

Las características de la placa Arduino Mega ADK empleada en este proyecto son las correspondientes a la revisión 3, que incluye las siguientes novedades, indicadas por el fabricante [6]:

- Pinout: Añadidos los pines SDA y SCL, que se encuentran junto al pin AREF y dos nuevos pines situados cerca del pin RESET, el IOREF que permite a los escudos adaptarse al voltaje facilitado por la placa. En un futuro, los escudos serán compatibles con la placa que usa el AVR, que opera con 5V y con Arduino Due que opera con 3,3 V. El segundo es un pin no conectado, únicamente reservado para usos futuros.

- Circuito de reset más fuerte.

La siguiente tabla muestra las especificaciones técnicas de la placa, recogidas por el fabricante:

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
USB Host Chip	MAX3421E
Length	101.52 mm
Width	53.3 mm
Weight	36 g

Figura 16. Especificaciones técnicas Arduino MEGA ADK

El Arduino MEGA ADK puede ser alimentado a través de la conexión USB o mediante una fuente de alimentación externa. La fuente de alimentación se selecciona automáticamente en caso de estar ambas presentes. En este caso, se ha empleado una fuente en lugar de la conexión USB.

Los pines de alimentación son los siguientes:

- VIN: Pin utilizado para alimentar el Arduino cuando se utiliza una fuente de alimentación externa (en vez de la conexión USB de 5V u otra fuente regulada).

- 5V: Este pin proporciona una salida de 5V regulada por la placa. El suministro de tensión a través de los pines 5V o 3,3V no pasa por el regulador interno de la placa, lo que podría ocasionar daños.

- 3,3V: Pin que proporciona una tensión de 3,3V controlados por el regulador de la placa. La máxima corriente es de 50mA.

- GND: Pines de tierra.

- IOREF: Proporciona la referencia de tensión con la que opera un micro controlador. Un escudo configurado puede leer el voltaje de dicho pin y seleccionar la fuente de alimentación adecuada o habilitar traductores de tensión en las salidas para trabajar con los 5V o los 3,3V.

Comunicación

El Arduino MEGA ADK tiene numerosas facilidades para comunicarse con un PC, otras placas de Arduino, u otros microcontroladores. ATmega2560 provee cuatro hardware UARTs para comunicación serie TTL (5V). El ATmega8U2 provee un puerto com virtual para la conexión software con un PC (Windows necesita un archivo .inf, pero OSX y Linux reconocen la placa como un puerto COM automáticamente. El software de Arduino incluye un monitor serie que permite enviar datos textuales simples desde la placa. Los leds RX y TX en la placa se iluminarán cuando se están transmitiendo los datos del chip y la conexión USB al ordenador, pero no para comunicaciones serie en pines 0 y 1.

Está disponible en la web una librería SoftwareSerial que permite la comunicación en cualquiera de los pines digitales del Arduino Mega ADK [7]. El código del programa se puede consultar en los Anexos 3 y 4.

Arduino UNO

Se trata de un microcontrolador basado en el ATmega328P. Tiene 14 pines de entrada/salida digitales (6 de ellos pueden emplearse como salidas PWM), 6 entradas analógicas, un cristal de 16 MHz, conexión USB, un cable Jack y un botón de reset.

Las características de la placa son similares a las del Arduino MEGA ADK, a diferencia de que esta última posee más pines de entrada/salida, siendo el Arduino UNO mucho más simple. Estos microcontroladores son los que colocaremos en cada una de las tres partes del robot para realizar la adquisición de los datos.

4.1.4. PC Toshiba Satellite i5

Para la realización del proyecto no se ha adquirido ningún equipo nuevo, sino que se ha empleado un ordenador portátil de uso doméstico con las siguientes características:

Toshiba Satellite i5
Tipo de producto: Portátil.
Color del producto: Negro, Grafito.
Factor de forma: Concha.
Frecuencia del procesador: 2,00 GHz
Familia de procesador: Intel Core i3-6xxx
Modelo del procesador: i5-6006U.
Memoria interna: 8 GB
Tipo de memoria interna: DDR3L-SDRAM
Memoria interna máxima: 8 GB.
Capacidad total de almacenaje: 500 GB
Unidad de almacenamiento: Unidad de disco duro
Capacidad de disco duro: 500 GB.
Diagonal de la pantalla: 39,6 cm (15.6")
Resolución de la pantalla: 1366 x 768 Pixeles Tipo HD: HD

En la Figura 17 se puede ver el resultado de la integración de los sensores y las cuatro placas Arduino sobre el robot manipulador.

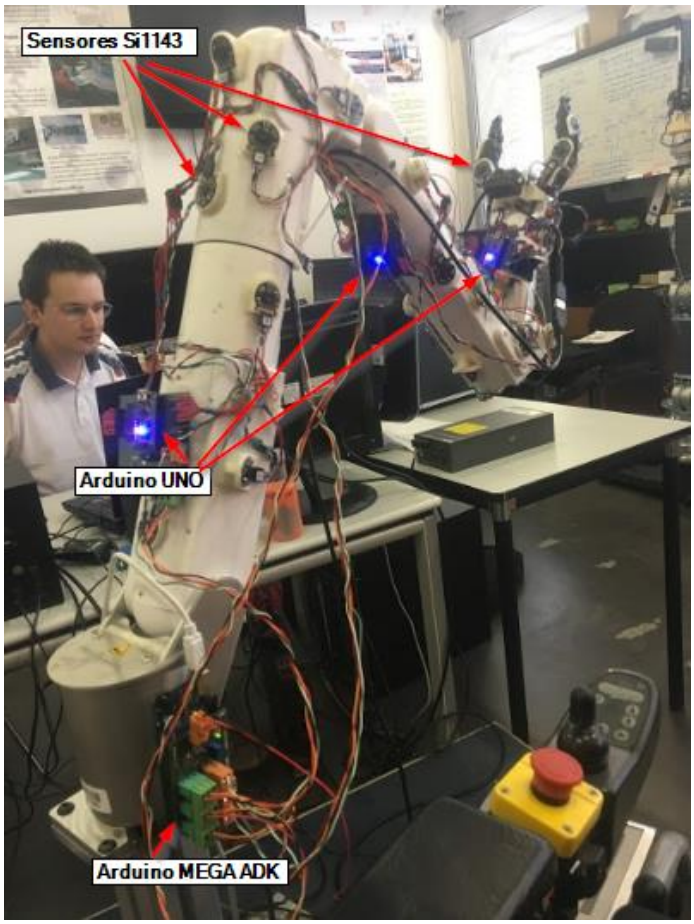

Figura 17. Integración elementos del sistema

4.2. Software

4.2.1. Ubuntu 16.04 LTS (XenialXerus)

Ubuntu es un Sistema Operativo que resulta de una combinación entre el Sistema Operativo GNU, desarrollado por la FSF (Free Software Foundation) y el núcleo (kernel) Linux, desarrollado por Linus Torvalds y la Linux Foundation. Es distribuido como software libre, e incluye su propio entorno de escritorio denominado Unity. Todo el código fuente de Ubuntu puede ser utilizado, modificado y redistribuido libremente por cualquier usuario, bajo los términos de la GPL (Licencia Pública General de GNU).

Figura 18. Ubuntu 16.04 LTS

Ubuntu es el principal Sistema Operativo empleado por los desarrolladores informáticos y de sistemas embebidos, ya que sus principales características son la robustez, versatilidad, amplia comunidad, libertad y gratuidad del código, lo que permite desarrollar sistemas seguros a bajo coste.

Durante el desarrollo de este proyecto se ha utilizado la versión de Ubuntu 16.04 LTS (XenialXerus), instalada en los ordenadores del Laboratorio Asistencial de la Universidad Carlos III de Madrid.

4.2.2. Arduino IDE

Para programar las tarjetas implementadas en nuestro robot, Arduino incluye un entorno interactivo de desarrollo (IDE), basado en Processing y Wiring, cuyo acceso es totalmente libre a los usuarios (Open Source).

El lenguaje de programación Arduino está basado en C/C++, y se apoya, además, en el uso de bibliotecas específicas para estos microcontroladores propias de Arduino.

Hay que tener en cuenta que Arduino, a diferencia de un ordenador convencional, no posee ni pantalla ni teclado, por lo que necesita un programa

externo que se ejecute en otro ordenador para poder incluir algún programa en sus placas. Este tipo de desarrollo de software, es lo que denominamos el IDE de Arduino. El programa será desarrollado en otro ordenador, a continuación será compilado y cargado en la placa a través del IDE, y finalmente el programa será ejecutado en la placa.

La conexión de las placas al ordenador externo se realiza mediante un cable USB, como cualquier otro periférico (teclado, impresora, etc). Para cargar cualquier programa en una placa, es necesario que esté conectada a un PC. Además, el cable USB también suministra energía, aunque está configurada para que pueda alimentarse mediante una fuente externa.

Para nuestro proyecto, realizaremos la conexión directamente a una fuente de alimentación, que suministrará 12 V a las placas.

Figura 19. Arduino IDE

4.2.3. YARP

YARP, acrónimo de Yet another Robot Platform, es un conjunto de protocolos, librerías y herramientas para mantener módulos y dispositivos desacoplados. Es considerado Middleware, ya que es un software que se sitúa entre el Sistema Operativo de nuestro ordenador y las aplicaciones que se ejecutan en él. Está escrito en C++ casi en su totalidad.

YARP es software abierto y gratuito, desarrollado por y para investigadores del campo de la robótica, sobretodo en robótica humanoide.

El funcionamiento de YARP se basa en un sistema de control de robots mediante una colección de programas que se comunican de forma peer-to-peer. Es decir, el intercambio de datos se realiza mediante una serie de nodos que se comportan como iguales entre sí. Cada elemento del sistema actúa simultáneamente como cliente y servidor, respecto a los demás nodos de la red. Esto permite el intercambio directo de información, de forma distribuida y en cualquier formato, entre los ordenadores interconectados.

Podemos tener acceso a todas las funciones, librerías y demás utilidades de YARP en la web oficial [8], donde encontramos, además, tutoriales para su aprendizaje desde cero.

Figura 20. YARP (Yet Another Robot Platform)

Para este proyecto se ha empleado la versión 2.3.20 de YARP, instalada en los ordenadores del Laboratorio de Robótica Asistencial de la Universidad Carlos III de Madrid.

5. Sistema propuesto

En este apartado se comentará dónde se enmarca este proyecto dentro de un sistema completo, para obtener una visión global del mismo y comprender su funcionalidad en la práctica. Además, se explicará el modo de interconexión entre aplicaciones y la composición de cada uno de los módulos.

En la Figura 21, se muestra un esquema de la aplicación de control del robot AMOR, donde queremos incluir nuestro módulo de detección de obstáculos, *Proximity Sensor Client*.

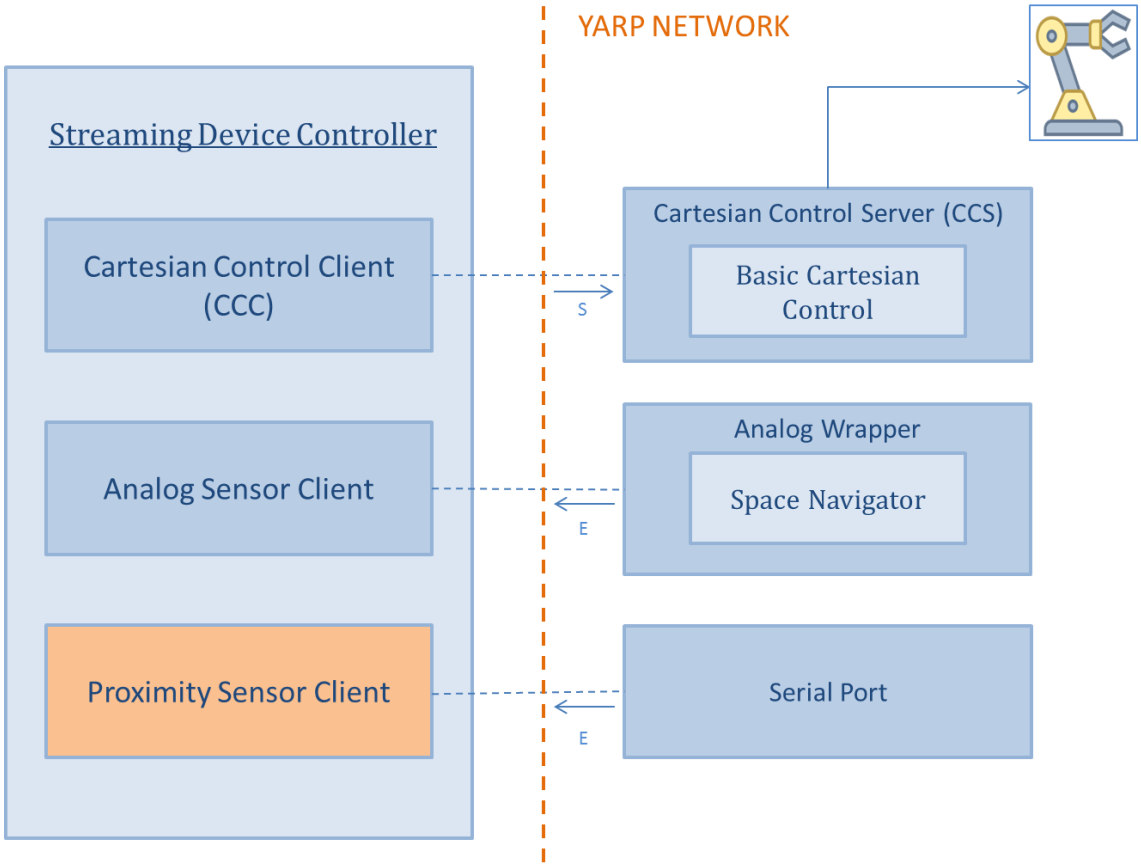

Figura 21. Esquema aplicaciones para el control de AMOR

La aplicación principal, que recibe el nombre de *Streaming Device Controller*, se ha desarrollado con el fin de controlar el brazo robótico mediante un joystick. Hay que destacar que esta aplicación no es exclusiva para el control de AMOR. De hecho, es compartida por varios proyectos que trabajan con diferentes robots en la universidad, como es el caso de ASIBOT y TEO.

Esta capacidad de adecuación entre sistemas es posible gracias a su arquitectura de diseño de aplicaciones, que sigue el modelo cliente/servidor. Este método de organización es muy habitual en sistemas distribuidos, y consiste en separar las funciones en dos partes: clientes y servidores.

Un cliente es un programa que usa los servicios proporcionados por otros programas. Aquellos que prestan los servicios se denominan servidores. El cliente realiza una petición de servicio y el servidor la lleva a cabo. Las funciones de servidor, en ocasiones necesitan algún tipo de gestión de recursos adicional, en el que un servidor sincroniza y gestiona el acceso a dichos recursos, y devuelve la petición al cliente con datos de respuesta o información de estado. Además, el cliente no necesita tener información acerca de cómo se gestionan los recursos. Si se modifica la base de datos empleada o los dispositivos conectados, debemos adaptar el servidor a nuestra nueva utilidad, pero no el cliente [9].

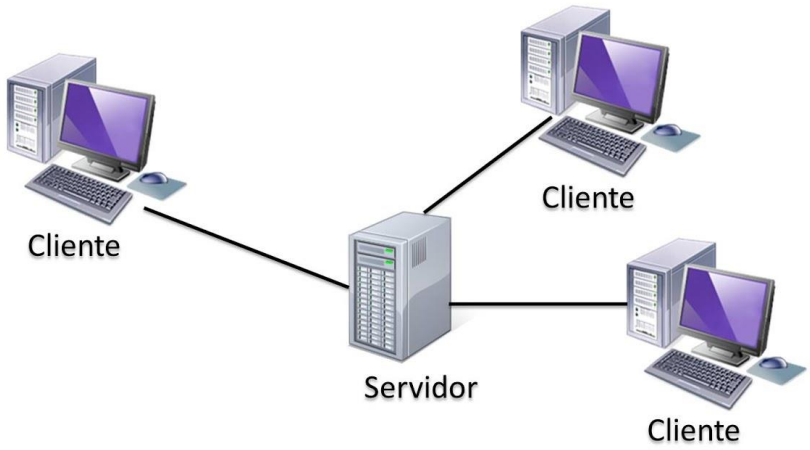

Figura 22. Esquema modelo de comunicación cliente/servidor

Los dos primeros módulos del esquema mostrado en la Figura 21, son los encargados del manejo del robot. El programa cliente, *Analog Sensor Client*, recibe la información del servidor *Space Navigator*, que contiene el código con la adquisición y configuración de los movimientos en los ejes X, Y, Z y las rotaciones rot X, rot Y, rot Z del joystick de control. Por otro lado, el programa cliente *Cartesian Control Client*, recibe los datos del servidor *Basic Cartesian Control*, que contiene el código que convierte los movimientos del joystick en movimientos articulares del robot. Es importante destacar que, aunque cada aplicación pueda ejecutarse desde un ordenador independiente, esta última con la cinemática inversa del robot debe utilizarse en el mismo PC donde conectemos el robot.

Otro elemento que debemos tener en cuenta es que, en los programas servidores, podemos observar que existen unos wrappers asociados a ellos. La traducción literal de wrapper es “envoltura”, y exactamente esta es la función que

desempeñan. Cuando trabajamos con Devices en YARP, es necesario contar con ellos para poder publicar los datos implementados en una interfaz a través de la red. Esto resulta útil, por ejemplo, cuando instalamos los drivers de un programa (servidor) en un ordenador que no es compatible. Con ayuda de los wrappers, podemos publicar a través de la red el resultado obtenido, y que el cliente, si se encuentra en una ubicación diferente, sea capaz de ejecutarlo.

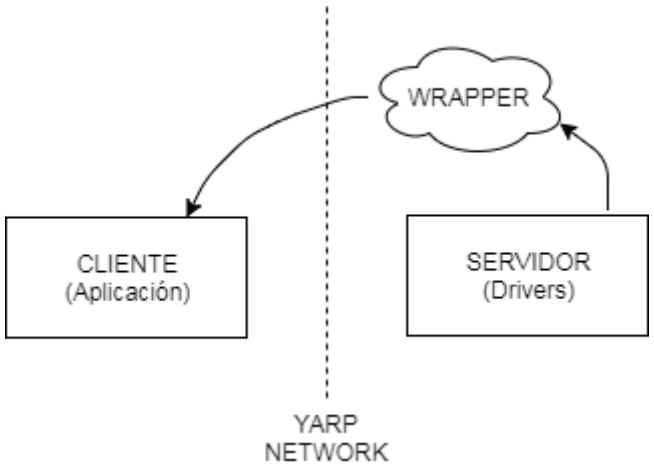

Figura 23. Esquema publicación de información con wrappers

Para completar esta aplicación, se ha planteado este proyecto para desarrollar un tercer módulo encargado de la seguridad a la hora de manejar el robot, mediante la detección de obstáculos y la prevención de colisiones. Esto lleva a destacar otra característica de la estructura de este programa: al estar dividida en módulos independientes, resulta sencillo e inmediato acoplar y desacoplar nuevas funcionalidades, o implementarlas en otros sistemas. A lo largo de la memoria se detalla cómo se ha implementado la configuración de este nuevo programa.

6. Desarrollo del proyecto

Como se ha comentado anteriormente, el proyecto se ha desarrollado en el Laboratorio de Robótica Asistencial de la Universidad Carlos III de Madrid, siendo AMOR el robot escogido para llevar a cabo el trabajo, cuya finalidad principal es prestar asistencia en tareas cotidianas a personas mayores o con algún tipo de discapacidad.

El principal objetivo de este proyecto es crear una aplicación que detecte obstáculos mediante sensores de proximidad, ya implementados en el robot en el desarrollo de un proyecto previo, realizar la adquisición y procesado de datos y configurar los actuadores para garantizar la seguridad según los niveles de alerta establecidos en base a las pruebas realizadas.

Para estas pruebas, se ha tomado un brazo humano como obstáculo, simulando que una persona se interpone en la trayectoria del robot, estableciendo una parada de emergencia para evitar colisiones, y después se ha realizado una caracterización para diferentes tipos de materiales.

En primer lugar, se ha realizado la adquisición de los datos de los sensores, junto con un procesado de los mismos y se han agrupado por zona del robot para tenerlos ordenados. Una vez hecho esto, se han numerado los sensores para tener identificada su posición en el robot y, a continuación, se han analizado el rango de medida y la cobertura total en la detección de obstáculos. Una vez finalizada la identificación de todos estos factores, se ha establecido una trayectoria fija para programar el control del robot y realizar las pruebas de manera sencilla, y por último, se ha desarrollado el módulo cliente/servidor con los sensores de proximidad para poder integrarlo en la aplicación global con el control por joystick.

Como ayuda en el desarrollo de las pruebas, se han configurado distintas funcionalidades gráficas que muestran los valores de los sensores y del estado del robot en cada momento, para poder asegurar que el sistema está funcionando de manera correcta.

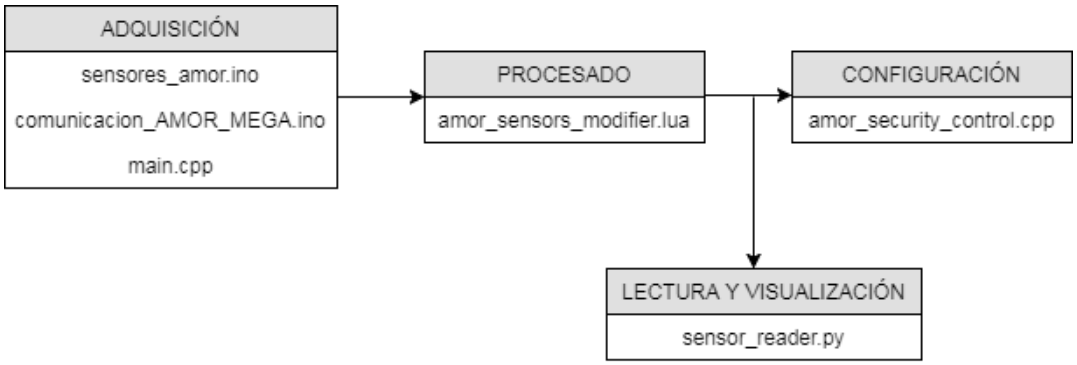

Figura 24. Archivos necesarios para las pruebas con trayectoria programada

6.1. Adquisición y procesamiento

6.1.1. Adquisición de los datos de los sensores

El primer paso para comenzar a trabajar en el desarrollo del proyecto, comienza con la investigación del funcionamiento del código ya elaborado en un proyecto previo para esta universidad, el cual no se encuentra publicado, por lo que no es posible su referenciación, que incorporaba la implementación del código en las placas Arduino para recibir las señales de los sensores. Una vez comprendido cómo se obtiene la respuesta de los mismos, se podrá comenzar a desarrollar el programa para procesar las señales y adaptarlas a nuestras necesidades.

Se pueden diferenciar dos partes en esta primera fase. En primer lugar, el código implementado en las placas Arduino de las tres partes del robot (brazo, antebrazo y garra), cuyo detalle puede consultarse en el archivo *sensores_amor.ino* del Anexo 1, que incluye la configuración de los pines de la placa, los puertos, y direcciones de memoria asignadas para su almacenamiento. En esta parte, se realizará la inicialización de todos los sensores y la lectura por puerto serie de los datos de cada uno de ellos. Para que sea posible esta comunicación, es necesario utilizar los archivos proporcionados por la comunidad de desarrolladores en Github para la configuración de la lectura por puertos serie. Si se desea ver en detalle el código, se deberán consultar los Anexos 3 y 4.

Al finalizar esta parte del programa, quedan identificados qué sensores se encuentran presentes y cuáles han fallado o se han deshabilitado, y se habrán asignado identificadores para los datos de cada parte del robot, [I, J, K], correspondientes a los sensores del brazo, antebrazo y garra, respectivamente. Cada una de estas partes, contará con un array de 16 dígitos que se completará con 1 y 0 si en función de si se encuentra o no presente el sensor en cada parte del robot.

Una vez esté configurada la lectura de datos de manera independiente en cada una de las tres placas, una cuarta placa será la encargada de agruparlos para poder tratarlos en un único bloque. Para ello, en la placa maestra se ha implementado el archivo *comunicación_AMOR_MEGA.ino* del Anexo 2, en el que se detallan cada uno de los pasos que realiza: identifica cada uno de los puertos serie abiertos para la comunicación correspondientes a cada una de las placas de las distintas partes del robot, espera a la inicialización de todas ellas, y comienza la lectura de los datos según los identificadores establecidos anteriormente, I, J, K. Con esto, tendremos todas las respuestas de los sensores organizados en un único bloque, y se transmitirán al usuario mediante un único puerto.

Una vez comprendido el funcionamiento de esta primera parte, podemos comenzar a desarrollar el trabajo propio de este proyecto.

6.1.2. Procesamiento de los datos adquiridos

El archivo con la configuración para el procesado de la señal, *amor_sensors_modifier.lua*, está compuesto por una serie de funciones que convierten los datos obtenidos de los sensores en valores fácilmente interpretables y adecuados para este trabajo. Los datos que se reciben en bruto, consisten en cadenas de caracteres en hexadecimal que no tienen ninguna utilidad, ya que no pueden traducirse en valores de distancia a un objeto, por lo que se ha realizado su conversión a formato decimal, entre otras adecuaciones.

El programa generado recibe los datos a través de la red de YARP, mediante un sistema de botellas. Las botellas en YARP consisten en una colección de objetos que pueden transmitirse de una manera portable. Los objetos se almacenan en una lista, a la que se puede acceder y añadir más objetos. La ventaja de utilizar este sistema, es que funciona como un array, sin importar que exista más de un tipo de datos en él.

Una vez se ha comprobado el estado de la conexión, comenzará la transmisión de los datos en las botellas a través del puerto /sensor_reader para, posteriormente, ser adaptados a las necesidades de nuestra aplicación.

A continuación, se enumeran las funciones principales que realiza el programa. Para ver el código completo, se puede consultar en el Anexo 5:

- Crea una instancia "SensorDataProcessor", que recibe como parámetros la tabla de datos con la salida de los sensores e inicializa todas las variables con las que se establecerá la configuración posterior (flags de error a cero, acumulador vacío, etc).

```
local SensorDataProcessor = {}

--
-- Creates a SensorDataProcessor instance.
--
-- @param processor Table of input parameters
--
-- @return SensorDataProcessor
--
SensorDataProcessor.new = function(self, processor)
local obj = {
accumulator = "",
dataReady = false,
currentSensorData = nil,
processor = processor,
stamp = 0
}
```

```
setmetatable(obj, self)
self.__index = self
return obj
end
```

- Establece una función con un acumulador que va almacenando de manera constante los datos recibidos por los sensores en una cadena de tipo string.

```
-- Consumes fetched data from sensors.
--
-- @param str String of new data to be appended to the accumulator
--
SensorDataProcessor.accept = function(self, str)
    str = str or ""
    self.accumulator = self.accumulator .. str
    self:tryConsume()
    end
```

- Setea un flag si los datos están listos para ser procesados, esto es, el sistema no presenta ningún fallo y las conexiones a la red y los puertos se han establecido correctamente. La función va analizando y procesando los datos que recibe del acumulador cuando detecta que el flag está activo.

```
-- Calls further processing methods if conditions are satisfied,
-- sets flag if data is ready to be forwarded to receiver.
--
SensorDataProcessor.tryConsume = function(self)
    if self.processor.evaluateCondition(self.accumulator) then
        self.currentSensorData, self.accumulator =
self.processor.process(self.accumulator)

        if self.currentSensorData then
            self.dataReady = true
            self.stamp = self.stamp + 1
        else
            print("message dropped")
        end
        end
```

- Convierte los datos de la tabla, inicialmente en formato hexadecimal expresados en una cadena de caracteres, en tipo decimal. Aquellos sensores que no presentan un dato válido (representado en hexadecimal con el carácter F), los establece a cero.

```
    for k, hexString in ipairs(thex) do
        local dec = tonumber(hexString, 16)
        if not dec then return nil end
        if dec == invalidValue then dec = 0 end
```

- Agrupa y ordena los datos ya procesados según los índices I, J, K establecidos para cada una de las partes del robot, y genera unos subíndices para identificar el número de sensor en cada parte. Cada sensor tiene como salida tres respuestas, correspondientes a los 3 LEDs.

```
PortMonitor.create = function(options)
    print("INITIALIZING AMOR SENSORS")
    -- TODO: read options from .ini file
    local meanProcessor = createMeanProcessor{
        parts = {"I", "J", "K"},
        subparts = {"X", "Y", "Z"},
        hexValues = 16,
        hexSize = 3
    }
    sensorDataProcessor = SensorDataProcessor:new(meanProcessor)
    return true;
    end
```

- Con los tres datos de cada sensor, realiza una media aritmética para obtener un único valor de respuesta para cada uno.

```
    -- Calculates arithmetic mean of input data.
    --
    -- @param varargs of input Numbers
    --
    -- @return Number as arithmetic mean of input data
    --
    local calculateMean = function(...)
        if select('#', ...) == 0 then return 0 end
        local sum = 0

        for i, value in ipairs({...}) do
            sum = sum + value
        end

        return math.floor(sum / select('#', ...))
        end
```

- Cuando una tabla está completa con una iteración, la resetea y comienza a generar una nueva. Antes de eliminar los datos de la tabla, los envía por el puerto de salida habilitado para la transmisión.

Una vez finalizada la parte de adquisición y procesado, se puede comenzar a trabajar con los datos para adaptarlos al objetivo de este proyecto.

6.1.3. Visualización gráfica de los datos procesados

Como ayuda al usuario, se ha generado un archivo adicional, *sensor_reader.py*, cuyo desarrollo se puede consultar en el Anexo 6 de esta memoria, que implementa una funcionalidad extra para poder visualizar los datos obtenidos por los sensores en una tabla, ordenados por zona del robot y número de sensor.

El programa generado recibe los datos a través de la red de YARP, mediante un sistema de botellas.

Una vez se ha comprobado el estado de la conexión, comenzará la transmisión de los datos en las botellas a través del puerto /sensor_reader. Se irá accediendo a cada uno de los elementos con la función b.get y se transformarán en datos de tipo double, para tener la lectura de la respuesta de los sensores en números decimales.

```
class DataProcessor(yarp.PortReader):
    def read(self, connection):
        print("in DataProcessor.read")
        if not connection.isValid():
            print("Connection shutting down")
            return False
        b = yarp.Bottle()
        print("Trying to read from connection")
        ok = b.read(connection)
        if not ok:
            print("Failed to read input")
            return False
        print("Received [%s]"% b.toString())
        sh1['text'] = 's1: %d' % b.get(0).asDouble()
        sh2['text'] = 's2: %d' % b.get(1).asDouble()
        sh3['text'] = 's3: %d' % b.get(2).asDouble()
```

Para finalizar, se ha implementado la función TKinter en el programa. TKinter es la GUI (Graphical User Interface) estándar para la visualización de datos de manera gráfica. Con esto, se genera una tabla dividida en tres secciones (brazo, antebrazo y mano), y se colocarán en orden las respuestas de los sensores para tener identificada su correspondencia.

Figura 25. Visualización de los datos con TKinter

De manera adicional, se ha ejecutado el comando *yarpscope*, propio de la red de YARP, que permite visualizar los datos en una gráfica X/Y, mostrando la respuesta de los sensores en función del tiempo. En la Figura 26 vemos un ejemplo de esta visualización.

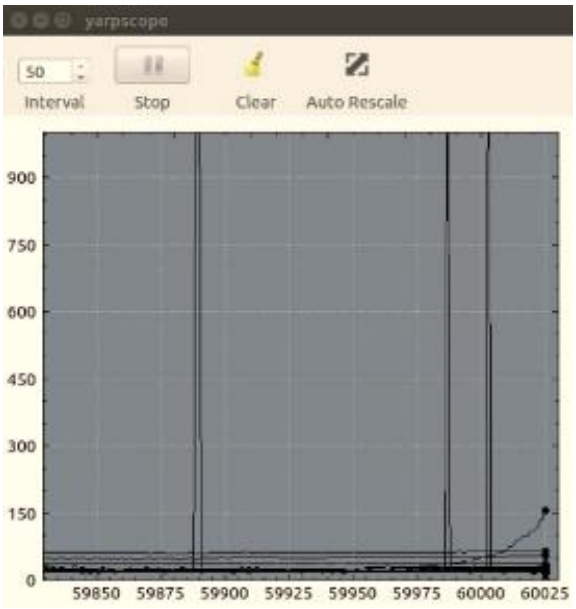

Figura 26. Visualización de los datos con yarpscope

6.2. Modelado de respuesta de los sensores

Para comenzar a trabajar con los sensores y definir cómo se desea que actúe el robot, es necesario analizar qué tipo de medidas se están obteniendo y su rango de detección (alcance).

En primer lugar, se han realizado una serie de medidas para observar los valores que se obtienen en función de la distancia del objeto al sensor, y se ha analizado la repetitividad de los mismos, de manera que se obtenga un resultado fiable para nuestro trabajo.

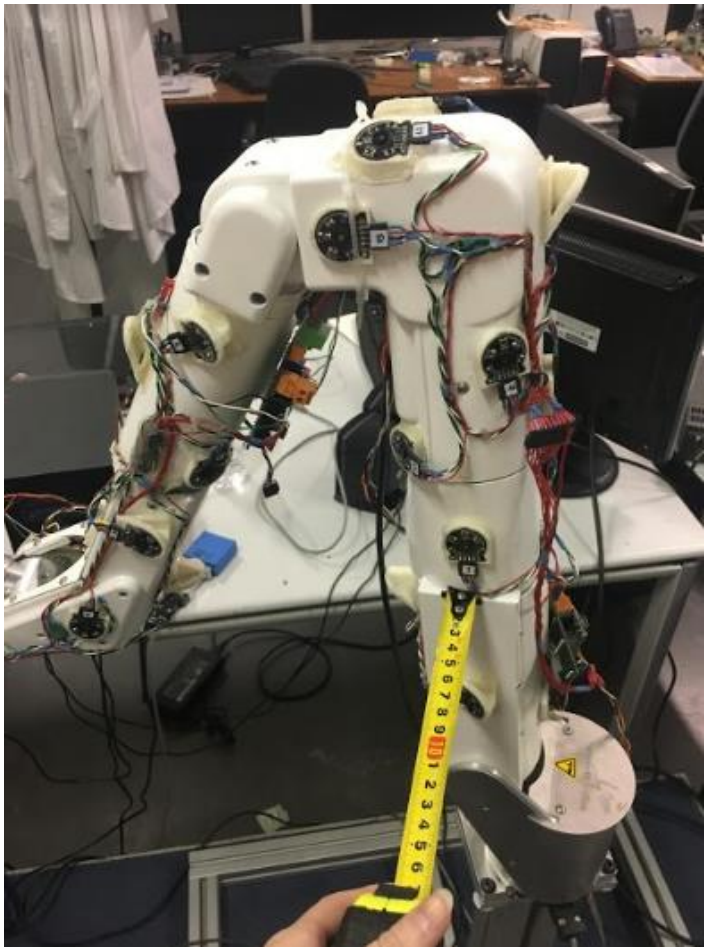

Figura 27. Medición de la respuesta de los sensores en el laboratorio

En la siguiente tabla se puede observar un ejemplo de algunos de los resultados obtenidos en varias pruebas de medida, empleando como obstáculo a detectar un brazo humano. Únicamente se muestra la respuesta de seis sensores, ya que los resultados son equivalentes para el resto de ellos:

Distancia (cm)	s6 brazo	s5 brazo	s10 antebrazo	s13 antebrazo	s9 mano	s15 mano
0	1400	1380	1400	1340	1460	1180
1	1300	1500	1500	1420	1554	1480
2	1200	1200	1100	1180	1250	1160
3	540	680	660	520	560	540
4	280	320	320	320	330	360
5	150	170	180	190	180	207
6	110	100	110	120	136	137
7	80	73	77	80	98	102
8	61	50	59	57	71	87
9	49	42	50	43	63	78
10	40	35	42	34	48	65
11	35	31	35	28	40	57
12	30	29	30	25	36	41
13	27	26	28	23	32	38
14	24	24	26	23	29	34
15	22	23	26	22	26	33
16	20	23	24	21	26	31
17	20	22	22	20	24	30
18	20	21	20	20	23	27
19	18	20	20	20	22	25
20	18	20	20	20	21	24

Figura 28. Tabla con respuesta de varios sensores ante un obstáculo

Con estos datos, se puede obtener el rango de medida de los sensores. En las hojas de características proporcionadas por el fabricante, se especificaba un alcance de hasta 50 cm en la detección de objetos. Sin embargo, en nuestro experimento se puede observar que, a partir de los 20 cm, las variaciones en la distancia son inapreciables en la respuesta de los sensores, por lo que no garantiza que el sistema sea capaz de detectar un obstáculo fuera de este alcance. Por tanto, se puede establecer como rango efectivo de medida de nuestros sensores [0, 20] cm.

En la Figura 28 se puede ver una gráfica con las respuestas de los sensores de la tabla anterior. Se puede concluir que el funcionamiento de los sensores es correcto, ya que coincide con la gráfica de respuesta indicada por el fabricante, incluida en el apartado 4.1.2. de esta memoria, y todos se comportan de la misma manera, ya que ninguno de ellos muestra una respuesta fuera del rango.

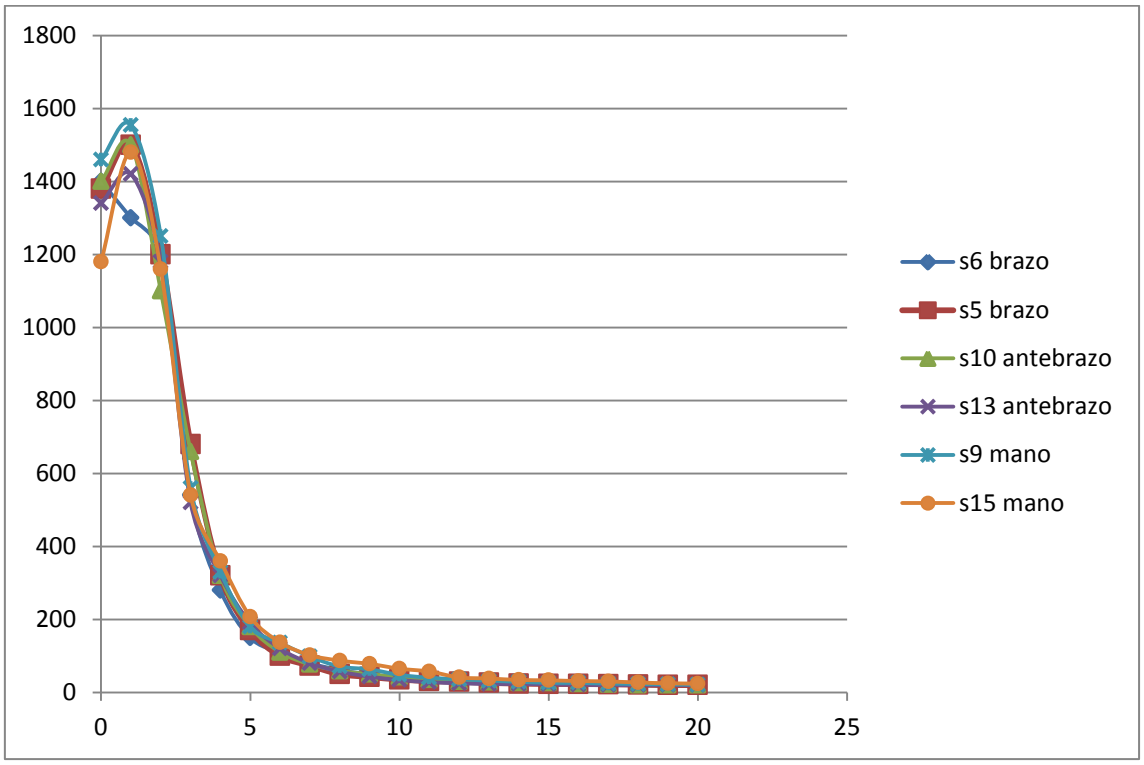

Figura 29. Gráfica con respuesta de los sensores tras las pruebas

De manera adicional, se han realizado pruebas con objetos de diferentes texturas y colores. De esta forma, se puede caracterizar la respuesta de los sensores para distintos tipos de materiales y analizar si podemos utilizar el mismo criterio para todos ellos en nuestros experimentos, o si es necesario especificar al principio de las pruebas con qué objetos estamos trabajando. En este caso, los elementos seleccionados se pueden encontrar en un entorno real de una vivienda. Como nuestro lugar de experimento es una cocina, se emplearán utensilios como latas, bricks, cacerolas, etc.

En la siguiente tabla se muestran las respuestas de los sensores con los diferentes objetos seleccionados:

Distancia (cm)	Brick	Cocacola (lata)	Madera	Botella (plástico)	Vaso (cristal)	Brazo humano	Camiseta blanca	Taza (ceramica)	Cacerola
0	1200	750	1200	95	500	1200	400	500	500
1	1500	1400	1700	300	250	1400	1900	1400	1400
2	1200	900	1300	130	180	800	1400	1000	1200
3	900	550	870	70	90	490	100	600	700
4	500	230	520	60	60	300	600	400	400
5	370	180	370	50	50	190	400	270	180
6	230	130	280	30	35	140	280	170	120
7	130	100	200	25	29	100	200	110	100
8	100	88	170	25	26	95	170	90	97
9	88	72	150	25	27	65	110	78	85
10	74	60	120	25	25	50	90	64	82
11	64	48	100	25	23	45	77	58	77
12	54	42	86	25	23	40	67	52	65
13	50	38	73	25	21	35	59	44	58
14	43	35	65	25	21	33	47	40	51
15	40	32	60	25	20	30	44	36	42
16	38	29	55	20	20	28	42	33	30
17	34	27	50	20	20	25	35	30	27
18	27	24	45	20	19	24	33	27	24
19	26	23	40	20	19	23	28	25	24
20	24	22	38	20	18	20	28	23	23

Figura 30. Respuesta de los sensores ante distintos materiales

Tras analizar los resultados obtenidos en este experimento, se ha observado que el los datos son muy variables en función del color y tipo de material con el que trabajemos, lo que puede afectar en mayor o menor medida a la detección de obstáculos. Los objetos transparentes como un vaso de cristal o una botella de plástico no son capaces de reflejar la luz emitida por los sensores, por lo que resulta difícil medir la distancia a estos. El en caso de objetos metálicos o de un color claro, reflejan perfectamente la luz del sensor, por lo que las medidas realizadas son claras y consistentes. En la Figura 30 se puede ver una gráfica con las respuestas de los sensores en función del material del obstáculo.

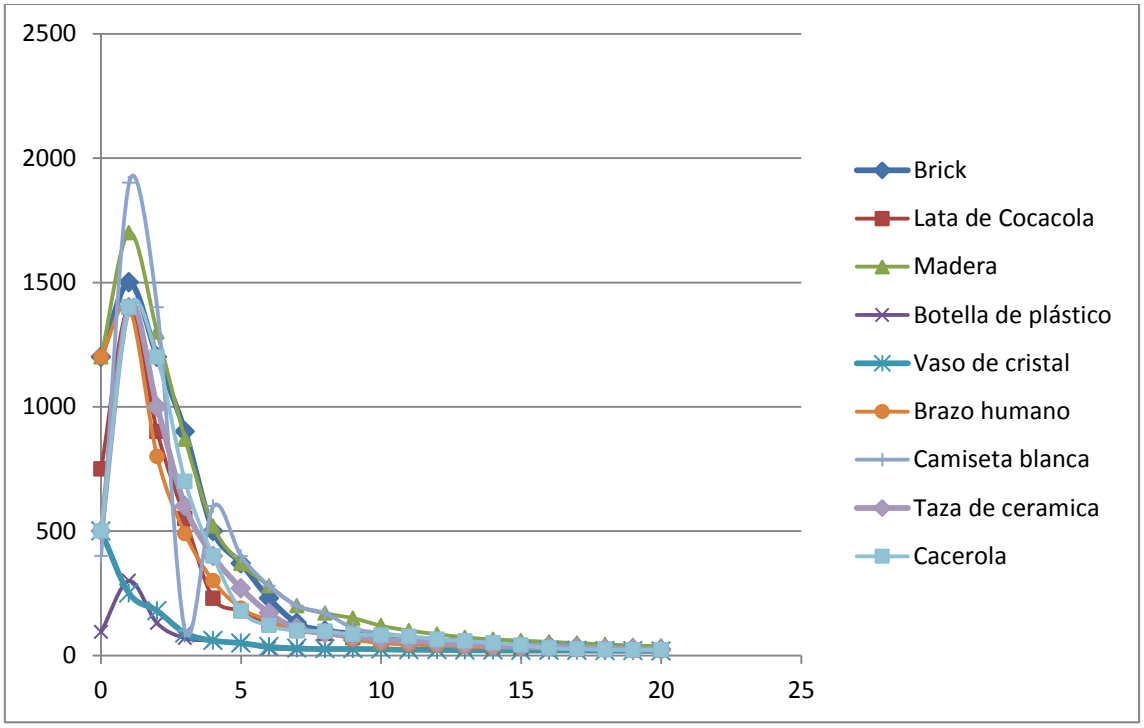

Figura 31. Gráfica con respuesta de los sensores según material del obstáculo

Con este análisis, se puede concluir que los mejores resultados obtenidos tienen lugar para objetos lisos y claros, con alta reflectancia, por lo que trataremos de adecuar nuestro ambiente de pruebas y nuestra estrategia de control a estos objetos para obtener unas medidas más fiables.

6.3. Prueba con trayectoria programada

Para las primeras pruebas de funcionamiento se ha creado un único archivo, básico y sencillo, que permita tener un buen control durante la realización del experimento, y asegurar que el robot funciona tal y como se desea.

En este programa, se ha incluido una trayectoria programada para el robot, de manera que tengamos localizados todos sus movimientos para simplificar las pruebas. Se ha establecido un punto inicial y otro final, en el que se sitúa una botella de Cocacola, y haremos que el robot se dirija hacia este objetivo. Cuando el sensor de la mano habilitado para este fin detecte su presencia, el robot cerrará la garra, atrapando el objeto.

Por otro lado, se ha establecido un nivel de alerta en el que, cuando alguno de los sensores del robot detecte que un obstáculo se encuentra a menos de 6 cm del mismo, detenga el movimiento del robot, reanudándose de forma automática cuando el obstáculo ha salido de su trayectoria.

Los niveles de alerta establecidos para el cierre de la garra y la parada de emergencia del robot se pueden consultar en la siguiente tabla:

Condición	Nivel de alerta	Acción requerida
Distancia obstáculo ≤ 6 cm	Flag=True	Controlled_stop
Distancia objetivo ≤ 1,5 cm	g_target=true	Amor_close_hand

Figura 32. Niveles de alerta en trayectoria programada

A continuación, se explicará en detalle en el funcionamiento del programa, cuyo código completo se incluye en el Anexo 9:

- Una vez se establezca la conexión con el robot y se hayan ejecutado los pasos previos de adquisición y procesado de la señal, nuestro programa recibirá los datos de los sensores a través del sistema de botellas de la red de YARP.

```
int main(int argc, char *argv[])
{

yarp::os::Network::init();

SensorReader sr;
sr.open("/sensor_reader");
sr.useCallback();


#ifdef __linux__
            g_hRobot = amor_connect((char *)"libeddriver.so", 0);
#endif // LINUX
```

```
if(g_hRobot == AMOR_INVALID_HANDLE)
return 1;                          // if AMOR connected
```

- Para configurar la trayectoria del robot, se han establecido en primer lugar los puntos inicial y final, con la posición en coordenadas cartesianas correspondientes a la base de la garra, donde se encuentra nuestro punto de referencia, respecto a la base del robot (origen de coordenadas), así como la posición angular de las articulaciones.

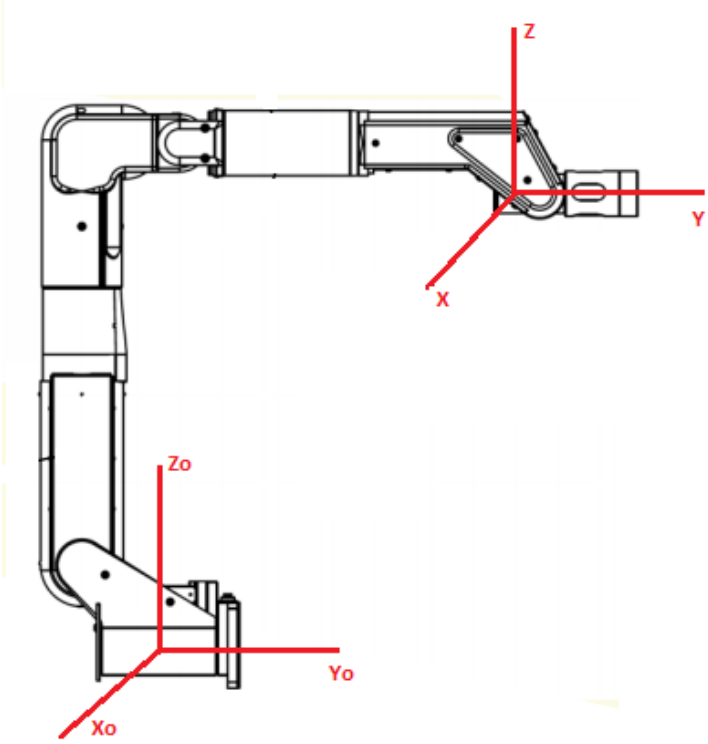

Figura 33. Sistemas de coordenadas en AMOR

```
const real INITIAL_CART_POSITION[AMOR_NUM_JOINTS] = {
      7.69, -419.72, 179.81, DEG2RAD(175.616), DEG2RAD(-90.216),
      DEG2RAD(-2.712) };

const real TARGET_CART_POSITION[AMOR_NUM_JOINTS] = {
      7.69, -637.74, 179.81, DEG2RAD(175.616), DEG2RAD(-90.216),
      DEG2RAD(-2.712) };
```

- A partir de la diferencia entre estos puntos, obtenemos el vector de la trayectoria que seguirá el robot. A continuación, se procede a normalizar el vector y a hacerlo unitario, ya que de esta manera, estableceremos la velocidad de movimiento. De manera adicional, se ha establecido una constante *c* que, multiplicada a la velocidad del robot, permitirá aumentarla o disminuirla cuando sea necesario.

```
//Hallar trayectoria del robot

   for(int i=0;i<3;i++)
   g_path[i]=TARGET_CART_POSITION[i] - g_cartPositions[i];

//Normalizar vector trayectoria

   g_Pnormalized=sqrt(g_path[0]*g_path[0]+g_path[1]*g_path[1]+g_path[2]*g
_path[2]);

//Obtener vector unitario

   if(g_Pnormalized>0)
   {for(int i=0;i<3;i++)
   {  g_path[i]/=g_Pnormalized;  }};

//Establecer velocidad del robot a partir de la trayectoria calculada
(VELOCIDAD NORMAL DE MOVIMIENTO, SIN OBSTÁCULOS)

   for(int i=0; i<3; i++)
         g_cartVelocity[i]=c*g_path[i];
   for(int i=0; i<6; i++)
         printf("%.2f ", g_cartVelocity[i]);
         printf("\n");
         yarp::os::Time::delay(5);
       if(amor_set_cartesian_velocities(g_hRobot, g_cartVelocity) !=
AMOR_SUCCESS)
         printf("%s\n", amor_error());
```

- Para la parte de detección de obstáculos, se han tenido en cuenta todos los sensores del robot, excepto el que se encuentra en la posición 14 del array, que se corresponde al sensor frontal de la garra, ya que este se ha destinado para la detección de la botella de Cocacola.

Figura 34. Array con sensores del robot

Se ha establecido una función que localice el valor máximo de todos estos sensores, y ese dato es el que se comparará con el establecido para activar nuestro nivel de alerta. Con esto, conseguiremos que el robot se detenga con que al menos un sensor detecte un obstáculo, sin tener que estar comparando cada uno de ellos con el valor límite.

Por defecto, el flag indicador de la presencia de obstáculos, *g_threshold*, se encontrará desactivado (false), y se activará (true) cuando detecte que un obstáculo se encuentra a 6 cm del robot.

```
class SensorReader : public yarp::os::BufferedPort<yarp::os::Bottle>
{
public:
    virtual void onRead(yarp::os::Bottle& b)
    {
       if(b.size()==0)
       return;
       if(b.get(14).asDouble()>g_hand_threshold && b.get(14).asDouble()<1000)
         {g_target=true;
          printf("Botella detectada\n");}
       double max=0;
       for(int i=0;i<b.size();i++)
       {if(b.get(i).asDouble()>max)
       max=b.get(i).asDouble();}
         printf("Current maximum sensor value: %f\n", max);
       if(max>g_threshold)
       flag=true;
       else
       flag=false;
```

- Para la parte que incluye la detección del objetivo, se ha habilitado únicamente el sensor central de la garra (posición 14). Cuando ésta se encuentre a 1,5 cm del objetivo, modificará el valor de la variable global *g_target*, y comenzará el cierre de la garra.

  Por otro lado, se ha de tener en cuenta que, si el robot sufre algún tipo de variación en su trayectoria, es posible que no alcance la posición final exacta, teniendo como consecuencia que no llegue a detectar su objetivo y nunca se detenga. Para evitar esto, se ha identificado una tolerancia alrededor del punto final de 1,5 cm, mediante la constante *g_hand_threshold*, que solventa este problema.

  De manera adicional, se han detectado valores erróneos en la respuesta de los sensores bajo luz fluorescente, teniendo como consecuencia unos picos de salida con un valor superior a 1000 durante un instante de tiempo, por lo que se incluirá un comparador en la fórmula que ignorará estos valores.

```
       if(b.get(14).asDouble() > g_hand_threshold &&
       b.get(14).asDouble() < 1000)
         {g_target=true;
          printf("Botella detectada\n");}
```

- Para la parte del control del robot, se ha recurrido a la API de AMOR [10], proporcionada por el fabricante. Las funciones y tipos de datos necesarios para el desarrollo de este programa han sido:

FUNCIÓN	EXPLICACIÓN
amor_connect	Abre/inicia una conexión con AMOR
amor_set_cartesian_positions	Establece las posiciones cartesianas del robot
amor_set_cartesian_velocities	Establece la velocidad cartesiana del robot
amor_get_cartesian_position	Obtiene la posición en coord. Cartesianas del robot
amor_controlled_stop	Realiza una parada controlada del robot
amor_release	Finaliza la conexión al robot
amor_close_hand	Cierre de la garra del robot

TYPEDEFS	EXPLICACIÓN
AMOR_HANDLE	Representa una conexión a la API de AMOR
AMOR_VECTOR7	Representación de un vector de 7 dimensiones

DEFINES	EXPLICACIÓN
AMOR_SUCCESS	Representa una ejecución satisfactoria de una función
AMOR_NUM_JOINTS	Cantidad de uniones en AMOR
AMOR_INVALID_HANDLE	Devuelto por la función amor_connect si falla la conexión

Una vez compilado y ejecutado el programa, se realizaron numerosas pruebas para comprobar su correcto funcionamiento, proporcionando unos resultados satisfactorios.

6.4. Creación del módulo de detección de obstáculos

Una buena práctica en programación consiste en dividir el programa en partes o módulos, de manera que podamos acoplar y desacoplar de forma independiente las distintas funcionalidades del mismo. Este modo de actuación se conoce como programación por capas, y permite un mejor mantenimiento del programa, ya que su principal ventaja es que el desarrollo puede llevarse a cabo en varios niveles, y en caso de tener que modificar una parte, únicamente afectará a un único módulo (abstracción), sin tener que revisar el código fuente del resto, ya que el sistema se ha reducido a un acoplamiento informático mediante una interfaz de paso de mensajes.

Esta división, además, presenta la ventaja de que facilita la aportación de otros programadores, ya que pueden integrar o modificar parte del código sin conocer cómo se ha desarrollado el programa entero, e incluso incluir nuevos módulos que se acoplen a nuestro programa.

Ventajas:

- Método de diseño muy flexible para mejorar la productividad de un programa.

- Reduce la documentación y mejora su manutención y modificación.

- Permite la reutilización de código para más de un programa.

- Reduce errores y simplifica la corrección de fallos al ser códigos independientes.

Desventajas:

- La separación por módulos requiere un mayor nivel y conocimiento de programación.

- Requiere mayor memoria y tiempo de ejecución.

La aplicación generada para la detección de obstáculos, denominada *Proximity Sensors Client,* se acoplará como módulo dentro de la aplicación global *Streaming Device Controller* explicada en el apartado anterior, donde se comentaba el sistema propuesto. Este módulo se implementa también con arquitectura cliente/servidor, y consta de las siguientes partes:

- Programa cliente: Es el que ejecutaremos dentro de la aplicación *Streaming Device Controller* para realizar la implementación del sistema de seguridad mediante reducciones en la velocidad de robot o paradas de emergencia según distintos niveles de alerta, a través de los métodos definidos en la clase que lleva el mismo nombre que la aplicación.

- Programa servidor: Contiene el código con la adquisición de los datos de los sensores directamente de las placas de Arduino, recibidas a través del puerto serie, que posteriormente se enviarán al cliente a través de la red de YARP.

En la figura 35 se muestra un esquema con los archivos generados para cada programa:

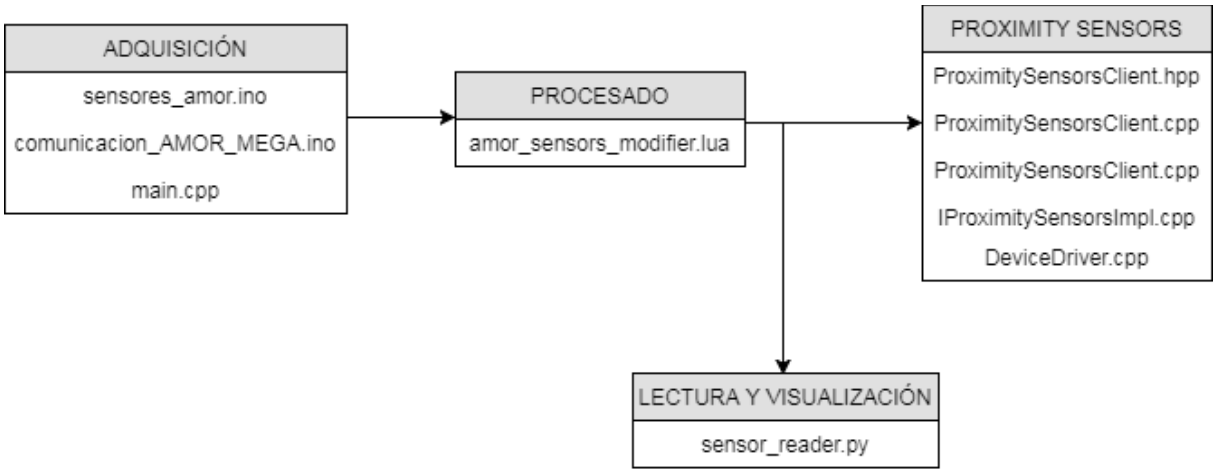

Figura 35. Archivos del módulo Proximity Sensors

En los siguientes apartados, se describen una a una las funcionalidades de cada bloque y su composición. Los módulos de adquisición, procesado y visualización de los datos ya se explicaron en el apartado anterior, y no se han modificado para esta parte, por lo que se comentará únicamente el bloque con el programa cliente.

6.4.1. Módulo Sensores de Proximidad

Este módulo está diseñado para la lectura de los sensores y el establecimiento de los niveles de alerta. Con esto, se podrá realizar la configuración en los actuadores del robot.

A continuación se muestra un diagrama de estados o UML, con las clases creadas para llevar a cabo este trabajo, con sus métodos y atributos, además de la relación de asociación entre ellas.

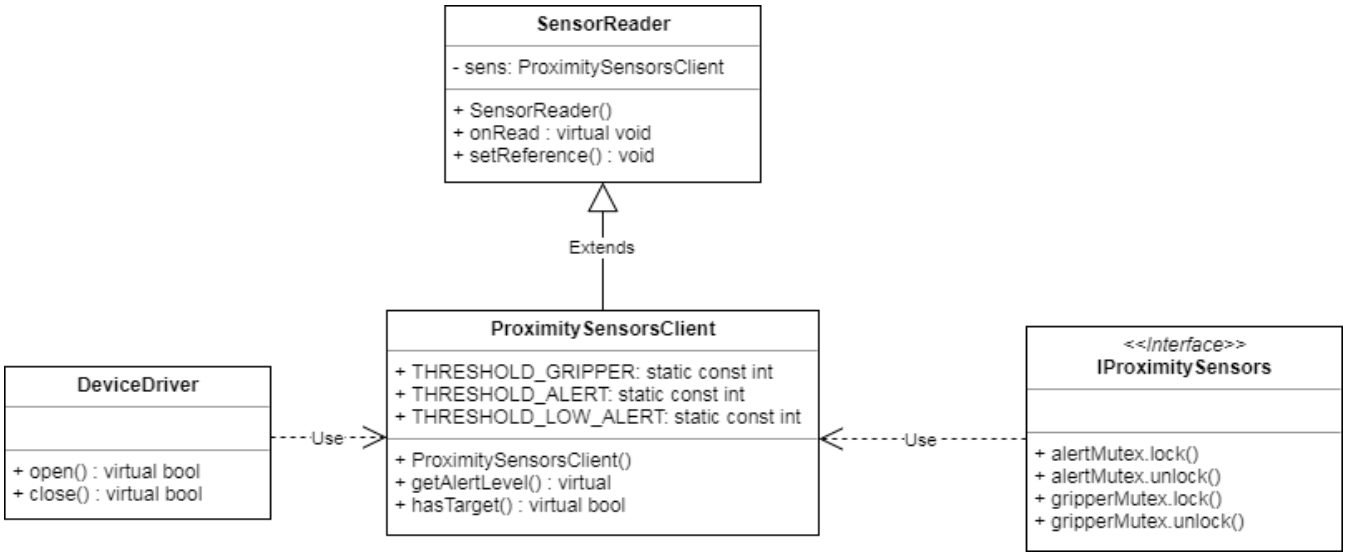

Figura 36. Diagrama UML con clases del módulo Proximity Sensors

Para completar este bloque, se han generado varios archivos que incluyen estas clases, cuyo contenido se explicará en los siguientes apartados.

ProximitySensorsClient.cpp

En este archivo se incluye la implementación de la clase *ProximitySensorsClient*, cuyos métodos y atributos son las funciones que permiten identificar los diferentes niveles de alerta.

```
const int roboticslab::ProximitySensorsClient::THRESHOLD_GRIPPER = 50;
const int roboticslab::ProximitySensorsClient::THRESHOLD_ALERT = 800;
const int roboticslab::ProximitySensorsClient::THRESHOLD_LOW_ALERT = 100;
```

Se puede observar que, para esta parte, se ha incluido un nuevo nivel de alerta. Inicialmente, el robot realizaba una parada controlada cuando un obstáculo se encontrase a menos de 6 cm. En esta ocasión, se ha reducido esta condición a los 2 cm de distancia, y se ha añadido un nivel de alerta bajo, que permitirá al robot disminuir su velocidad a la mitad cuando un obstáculo se encuentre a menos de 7 cm. En la siguiente tabla se explican las equivalencias para cada uno de ellos:

Condición	Nivel de alerta	Acción requerida
Distancia obstáculo ≤ 7 cm	THRESHOLD _LOW_ALERT	Reducción ½ velocidad
Distancia obstáculo ≤ 2 cm	THRESHOLD _ALERT	Controlled_stop
Distancia objetivo ≤ 1,5 cm	THRESHOLD_GRIPPER	Amor_close_hand

Figura 37. Niveles de alerta del módulo Proximity Sensors

Además de definir los niveles de alerta, la clase *ProximitySensorsClient* está configurada para recibir los datos de la red de YARP a través de las botellas. A partir de estos datos, se ha implementado una función que obtiene el máximo valor del sensor en cada una de ellas. Este valor será el que comparemos con nuestros niveles de alerta establecidos, y con la función definida para comprobar si el robot ha detectado un objeto con la garra. Para consultar en detalle el código, hay que acudir al Anexo 10. En la siguiente imagen se muestra la implementación de la clase *ProximitySensorsClient* con los atributos y métodos comentados.

```
void
roboticslab::ProximitySensorsClient::SensorReader::onRead(yarp::os::Bottle&
b)
{
    if (b.size() == 0)
        return;

    sens->gripperMutex.lock();
    if(b.get(14).asDouble() > THRESHOLD_GRIPPER && b.get(14).asDouble() <
1000)
         {sens->gripper=true;
          CD_INFO("Botella detectada\n");}
    else
       sens->gripper=false;
    sens->gripperMutex.unlock();

      double max=0;
      for(int i=0;i<b.size();i++)
      {if(b.get(i).asDouble()>max)
      max=b.get(i).asDouble();}
        CD_INFO("Current maximum sensor value: %f\n", max);
    sens->alertMutex.lock();
    if(max > THRESHOLD_ALERT)
    sens->alert = HIGH;
        else if (max > THRESHOLD_LOW_ALERT)
        {sens->alert = LOW;}
            else
            sens->alert = ZERO;
```

Para su interconexión con el resto de archivos, al comienzo del mismo deberá incluir a *ProximitySensorsClient.hpp*, donde se encuentran definidas las funciones y atributos empleadas en este código.

ProximitySensorsClient.hpp

En este archivo se incluye la declaración de la clase *ProximitySensorClient* que contiene el código con la configuración de los niveles de alerta para el control de obstáculos, y la detección del objetivo en la garra del robot.

En primer lugar, se ha incluido la declaración de la red de YARP, para poder tener acceso a los datos enviados por los sensores. Además, ha sido necesario definir los puertos usados para la recepción de los datos, así como los archivos encargados de la adquisición y procesado de los mismos.

```
#ifndef __PROXIMITY_SENSORS_HPP__
#define __PROXIMITY_SENSORS_HPP__

#include <yarp/os/all.h>
#include <yarp/dev/Drivers.h>
#include <yarp/dev/PolyDriver.h>

#include "IProximitySensors.h"

#define DEFAULT_LOCAL "/sensor_reader"
#define DEFAULT_REMOTE "/serial/out"

#define DEFAULT_PORTMONITOR_TYPE "lua"
#define DEFAULT_PORTMONITOR_CONTEXT "sensors"
#define DEFAULT_PORTMONITOR_FILE "amor_sensors_modifier"
```

Una vez hecho esto, se ha procedido a definir la clase *ProximitySensorsClient*, cuya implementación se encuentra en el archivo .cpp con el mismo nombre. Como podemos ver en el Anexo 11, esta clase tiene como métodos públicos la obtención de la respuesta según los niveles de alerta, así como la inicialización de las variables correspondientes, y como métodos protegidos los heredados de la clase *SensorReader* para la apertura de puertos y lectura de datos recibidos de la red de YARP, propias de esta clase.

```
class ProximitySensorsClient : public yarp::dev::DeviceDriver, public
IProximitySensors
{
    public:

        ProximitySensorsClient() : alert(ZERO), gripper(false)
        {}

        // -------- IProximitySensors declarations.
             // --------Implementation in IProximitySensorsImpl.cpp --------

        virtual alert_level getAlertLevel();
        virtual bool hasTarget();

        // -------- DeviceDriver declarations.
             // -------- Implementation in IDeviceImpl.cpp --------

        virtual bool open(yarp::os::Searchable& config);

        /**
        * Close the DeviceDriver.
        * @return true/false on success/failure.
```

```
        */
        virtual bool close();

        static const int THRESHOLD_GRIPPER;
        static const int THRESHOLD_ALERT;
        static const int THRESHOLD_LOW_ALERT;

    protected:

        class SensorReader : public yarp::os::BufferedPort<yarp::os::Bottle>
        {
        public:
            SensorReader() : sens(NULL) {}
            virtual void onRead(yarp::os::Bottle& b);
            void setReference(ProximitySensorsClient * sens)
            {
                this->sens = sens;
            }
        private:
            ProximitySensorsClient * sens;
        };

        SensorReader sr;
        alert_level alert;
        bool gripper;
        yarp::os::Mutex alertMutex, gripperMutex;

};
```

IProximitySensorsImpl.cpp

Este archivo compone la interfaz de nuestro programa, es decir, contiene la lista de métodos que hemos definido para permitir a un cliente usar nuestra librería. Será el archivo que conectará y hará que el resto de módulos reconozcan nuestras variables.

En primer lugar, se deben incluir en las cabeceras el archivo *ProximitySensorsClient.hpp*, ya que es el programa usado por el cliente en este proyecto. A continuación definiremos cada una de las clases utilizadas con sus métodos y atributos correspondientes.

La clase *IProximitySensors* es la propia de la interfaz, que tiene como finalidad realizar una copia de los valores indicados en los niveles de alerta. Estos niveles los obtendrá por herencia de los métodos *getAlertLevel()* y *hasTarget()*, propios de la clase *ProximitySensorClient*, En la siguiente imagen podemos ver su implementación.

```
roboticslab::IProximitySensors::alert_level
roboticslab::ProximitySensorsClient::getAlertLevel()
```

```
{
    alertMutex.lock();
    alert_level alert_copy = alert;
    alertMutex.unlock();
    return alert_copy;
}

bool roboticslab::ProximitySensorsClient::hasTarget()
{
    gripperMutex.lock();
    bool gripper_copy = gripper;
    gripperMutex.unlock();
    return gripper_copy;
}
```

6.4.2. Configuración del plug-in para el módulo sensores.

Un plug-in es una aplicación o programa que se relaciona con otra para añadirle una nueva funcionalidad, generalmente específica. Esta nueva aplicación se ejecuta por la aplicación principal, interactuando entre sí mediante una interfaz de aplicaciones [11].

Para que nuestro programa de sensores de proximidad sea configurado como un plug in, y pueda ejecutarse por la aplicación principal *Streaming Device Controller*, a través de la interfaz definida en el apartado anterior, YARP permite realizar su implementación como un device o dispositivo, a través de la clase *DeviceDriver* ya preconfigurada (ver Anexo 8).

Un Device Driver o controlador de dispositivo es un programa informático que permite al sistema operativo interaccionar con un periférico, realizando una abstracción del hardware y proporcionando una interfaz estandarizada para utilizar el dispositivo. Las funciones que un Device Driver maneja son las siguientes:

- Realiza una lectura abstracta e independiente del sistema, que procede de nuestro dispositivo, en nuestro caso los sensores de proximidad.

- Inicializa el dispositivo.

- Puede manejar eventos o requisitos específicos para el dispositivo.

- Comprueba que los parámetros de entrada sean válidos para nuestro sistema.

- Traduce datos de entrada válidos de términos abstractos a específicos para nuestro programa.

- Comprueba que el dispositivo esté en uso, por ejemplo, los bits de estado.

- Controla el dispositivo mediante una secuencia interna de comandos determinada por el driver para cada caso específico.

Para nuestro proyecto específico, el código con la programación del Device Driver realiza las siguientes acciones:

1) Apertura del programa e inicialización del sistema.

2) Comprobación de la configuración de los puertos.

3) Establece un callback para una adquisición constante de los datos.

4) Cierre del programa y deshabilitación del callback.

Además, debemos declarar en el código el archivo *ProximitySensorsClient.hpp*, que incluye la definición de las funciones propias que hemos elaborado para nuestro programa cliente, implementadas en el cpp con el mismo nombre, *ProximitySensorsClient.cpp*, de manera que exista una interconexión entre archivos para la identificación de clases y variables.

MUTEX

A la hora de ejecutar nuestro programa, hay que tener en cuenta que existen varios hilos abiertos en la comunicación. Por ejemplo, encontramos el hilo principal donde se inicia el programa, y un hilo secundario con el callback, o la recepción de datos a través de las botellas de YARP. En los hilos se realiza, de manera constante en el programa, una lectura y escritura de datos, para comprobar el estado de los flags.

Para evitar errores en el programa, y que no exista una concurrencia en la lectura/escritura, se establece una protección mediante el uso de un semáforo, empleando para ello la función Mutex. Los Mutex serán el mecanismo que nos ayude en la sincronización para proteger una sección crítica en nuestro código. En nuestro programa, inicializaremos el valor del Mutex a 1, estableciendo ese como el número de hilos que pueden acceder cada vez. Activaremos y bloquearemos el semáforo al comienzo y final de cada bloque de código que queramos realizar con una secuencia única, mediante las siguientes funciones:

- Mutex.lock: Bloquea el Mutex si no lo está empleando ninguna otra parte del código. Si alguien tiene bloqueado el Mutex el proceso espera hasta que el proceso que lo tiene bloqueado lo libera.

- Mutex.unlock: Libera el Mutex.

En la siguiente imagen se muestra un esquema con el funcionamiento del Mutex en la sincronización de dos hilos.

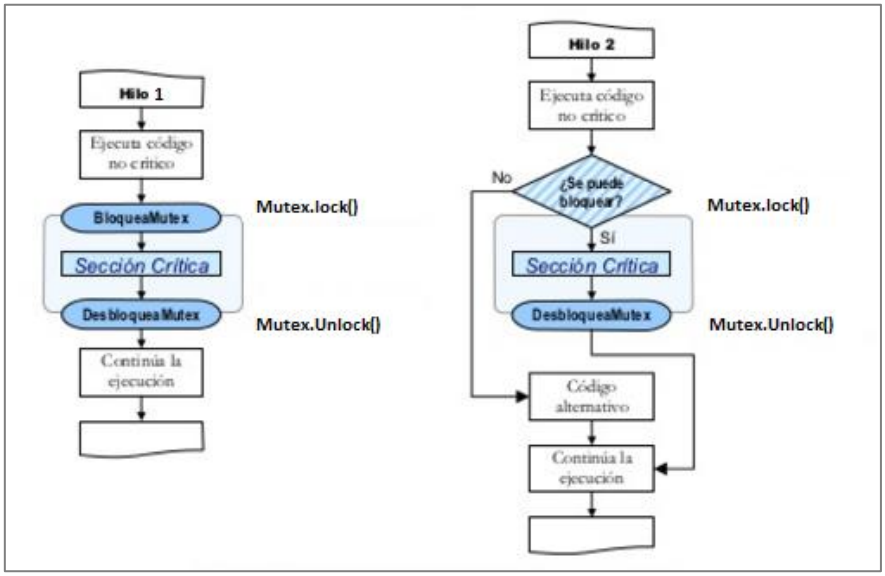

Figura 38. Esquema funcionamiento del MUTEX

7. Análisis y conclusiones

Para concluir el proyecto, se realizará un análisis de su desarrollo, incluyendo las dificultades que se han presentado durante su realización, hasta la consecución del objetivo con de manera satisfactoria. Por último, se incluirá un apartado con las conclusiones obtenidas tras finalizar el estudio.

7.1. Análisis del desarrollo del TFG

A continuación se expondrán las principales ventajas e inconvenientes encontrados durante el desarrollo del proyecto:

Ventajas

- Adaptabilidad a distintos sistemas: Al dividir el programa en módulos independientes, hace posible su adaptación a otro robot con características similares, y permite además la integración de más elementos al sistema, como por ejemplo una cámara RGB-D u otro tipo de sensores. Simplemente sería necesario crear módulo específico para este nuevo elemento y vincular esta nueva interfaz a la nuestra, para que puedan funcionar de manera conjunta.

- Facilidad en el desarrollo colaborativo: de nuevo, al estar el código dividido de manera independiente, resulta más sencillo a otros programadores realizar modificaciones o ampliar funcionalidades en el código sin necesidad de acceder o comprender el funcionamiento del resto de módulos.

- Integración en un sistema real: desde el inicio del proyecto, tanto la programación como las pruebas de usuario se han desarrollado en un entorno real (la cocina de una vivienda instalada en el laboratorio), y directamente sobre el robot, en lugar de mediante simulaciones, lo que asegura un funcionamiento correcto a la hora de probarlo con un paciente.

Figura 39. Entorno de realización del proyecto

- Fácil utilización una vez implementado: el desarrollo del proyecto implica un conocimiento alto de programación, para configurar el funcionamiento de cada uno de los módulos, y su posterior integración en un mismo sistema. Sin embargo, una vez implementado, su uso resulta muy sencillo de cara al usuario, ya que únicamente deberá encargarse de manejar el joystick para dirigir los movimientos del robot, quedando la programación de la seguridad en un segundo plano destinado exclusivamente a los programadores.

Inconvenientes

- Orden de ejecución estricto: durante el desarrollo ha sido necesario seguir un orden específico, ya que para avanzar con las tareas era necesario que las etapas anteriores funcionasen correctamente. Por ejemplo, resulta imposible configurar los niveles de alerta o los actuadores del robot, si no contamos con la respuesta de los sensores ante la presencia de un obstáculo.

- Múltiples pruebas de caracterización: para poder abarcar la gran cantidad de obstáculos que pueden presentarse en un entorno real, es necesario realizar numerosas pruebas con objetos de distinto material, color, etc., y bajo distintas condiciones de luz, ya que la respuesta de los sensores varía en cada una de ellas.

- Limitación en los movimientos del robot: durante el control por joystick del brazo robótico, se detectó que, al realizar movimientos en los que sus partes quedan muy contraídas, el robot es incapaz de recuperar la posición y es necesario reiniciar el sistema, enviando un error en la cinemática inversa. Para las pruebas, se evitó realizar este tipo de movimientos, y se planteará

en un siguiente punto de la memoria la mejora de este aspecto en un trabajo futuro.

- Inestabilidad de la señal bajo ciertas condiciones ambientales: durante las pruebas en el laboratorio se detectó que, bajo una luz fluorescente, en ciertas ocasiones aparecían unos picos en la salida de los sensores, que mostraban valores muy elevados que podían llegar a interferir en la detección de obstáculos. Este punto no se pudo llegar a solventar, sin embargo, en nuestra aplicación no supuso ningún problema ya que, al tratarse de un pico instantáneo, era indetectable para nuestro programa.

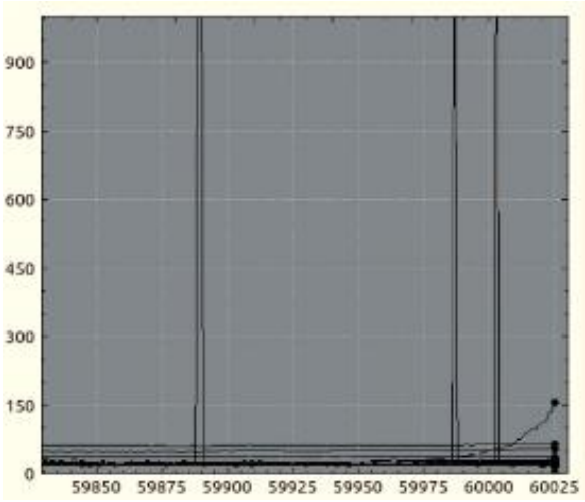

Figura 40. Picos en la señal de los sensores

- Cortocircuito en la parte de la mano: durante los experimentos se observó que los sensores de la garra presentaban errores de medida en algunas ocasiones. Tras revisar el robot, se detectó que esta parte, al ser metálica (ver Figura 41), daba lugar a un cortocircuito al entrar en contacto el cableado interno del robot y los sensores acoplados por la parte externa. Este problema se solventó situando un elemento aislante entre los sensores y el robot.

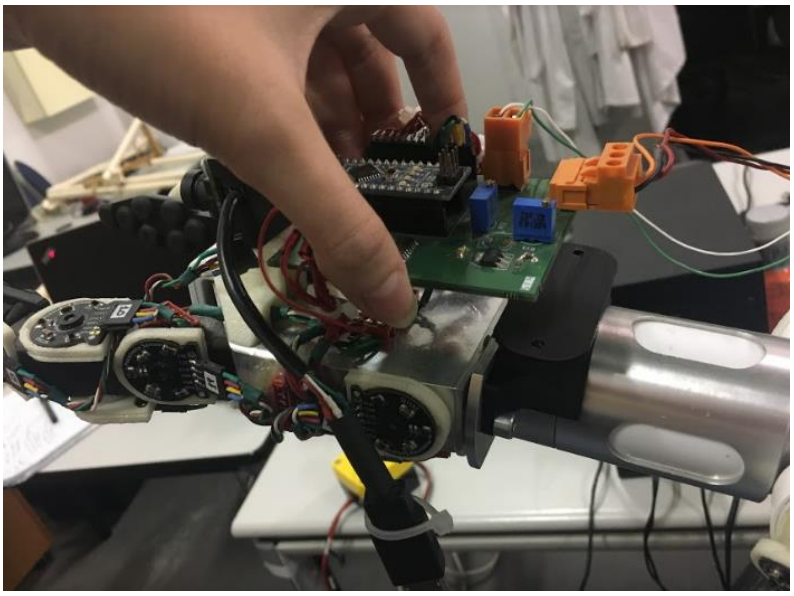

Figura 41. Pieza aislante diseñada para evitar cortocircuitos

- Localización del entorno de pruebas: para llevar a cabo el proyecto, ha sido necesario acudir al Parque Científico de la Universidad Carlos III de Madrid, ya que el robot se encontraba en el Laboratorio de Robótica Asistencial. Únicamente se han podido realizar a distancia una serie de pruebas con un sensor aislado, recopilación de información para comprender el funcionamiento de los sensores, el estudio del lenguaje de programación en C++ y la API de AMOR, y la redacción de la memoria.

- Utilización de Linux para el desarrollo del proyecto: este Sistema Operativo es el favorito para los programadores debido a su robustez, versatilidad, amplia comunidad, libertad y gratuidad del código, que permite desarrollar sistemas seguros a bajo coste. Sin embargo, fuera de este ámbito no acostumbramos a usar este sistema, por lo que resulta complejo y poco intuitivo comenzar a trabajar con él.

7.2. Conclusiones tras finalizar el proyecto

El proyecto tenía como objetivo principal la revisión y puesta en marcha de sensores de proximidad sobre el robot AMOR, para desarrollar un sistema de seguridad de detección de obstáculos. Una vez logrados los objetivos, el programa debe evitar que el robot colisione con los posibles objetos que se encuentren en su trayectoria y no causar daños en el entorno o en el usuario.

En primer lugar, se ha garantizado la correcta adquisición de datos por parte de los sensores. Para ello, se ha revisado tanto el cableado del bus i^2C como

posibles fallos de conexión con el cuerpo del robot, por ejemplo, cortocircuitos con partes metálicas.

Por otro lado, se ha caracterizado la respuesta de los sensores de forma experimental para distintos objetos y materiales. Este estudio preliminar ha mostrado que los datos pueden variar en gran medida en función del color y material del objeto a detectar, siendo los objetos metálicos o de color claro los que presentan mayor fiabilidad en las medidas.

También se ha podido comprobar que el rango de medida de los sensores indicado por el fabricante no se cumple en ningún de las pruebas. Para solventar este problema, se debería aumentar la ganancia de los sensores reprogramando el código en los Arduinos, solución que se planteará en el apartado Trabajos Futuros.

Para implementar la solución propuesta en este trabajo, se han desarrollado una serie de módulos independientes separados por su funcionalidad: adquisición, lectura, procesado, y control de los actuadores. Esta arquitectura distribuida facilita su ampliación o el desarrollo de mejoras de cara a trabajos futuros. También se ha incluido el programa completo en la plataforma online Github, de manera que el código esté disponible para otros programadores y puedan trabajar sobre él, manteniendo un historial de versiones.

A lo largo del desarrollo del proyecto, se ha trabajado con el lenguaje de programación C++ en el sistema operativo Linux. También ha sido necesario introducirse en el lenguaje de programación Arduino para conocer el funcionamiento de los sensores, ya que el trabajo previo del que ha partido este proyecto se realizó con este lenguaje. Para el control del robot, también ha sido necesario recurrir a la API de AMOR, con funciones específicas prediseñadas por el fabricante para el control de los actuadores.

El desarrollo de este proyecto ha supuesto un reto desde el principio, ya que mis conocimientos en lenguaje de programación se limitaban a C++ en un nivel medio, y ha requerido un gran trabajo de investigación y consulta en diferentes foros y webs de desarrolladores. El tener la oportunidad de realizar este trabajo, me ha permitido adquirir un elevado conocimiento de la materia, así como conocer las posibilidades que tiene el sector de la robótica en el mundo de la medicina, como en otras múltiples aplicaciones, teniendo grandes perspectivas de avance y mejora.

Finalmente, tras meses de dedicación y esfuerzo, se ha conseguido lograr el objetivo propuesto, logrando garantizar la seguridad en la utilización de un robot asistencial que mejorará la calidad de vida de las personas.

8. Posibles mejoras y trabajos futuros

El objetivo marcado para este proyecto se ha cumplido de manera satisfactoria, sin embargo, se pueden plantear una serie de mejoras en el programa, así como la resolución de algunos fallos encontrados que no se han podido solventar, aunque no hayan afectado a nuestro trabajo. Además, durante el desarrollo del proyecto, han ido surgiendo propuestas de trabajos futuros para perfeccionar su funcionamiento, que se indicarán en los siguientes puntos:

- Incluir un filtro en el código de los Arduinos para corregir los picos en la respuesta de los sensores bajo la luz de los fluorescentes.

- Aumentar la ganancia de los sensores para ampliar el rango de medida.

- Generar una trayectoria alternativa cuando el robot detecte un obstáculo, para que pueda evitarlo sin detenerse.

- Realizar una configuración de manera efectiva para que identifique el tipo de objeto que está detectando, y detecte la distancia al objeto.

- Incluir una opción al inicio del programa que nos permita seleccionar el entorno en el que se están haciendo las pruebas, para adaptarlo a las condiciones específicas del ambiente y de los objetos o nivel de luz que podamos encontrar.

- Añadir un sistema de control del robot por voz, en lugar de utilizar un joystick, dirigido a pacientes que presenten falta de movilidad en la mano.

9. Marco regulador

A lo largo de este apartado se realizará un análisis de la legislación aplicable sobre la implementación descrita en el trabajo (riesgos, responsabilidades profesionales, responsabilidades éticas, riesgos laborales, privacidad y seguridad, etc.).

Por otro lado, se analizarán los estándares técnicos sobre la robótica asistencial y el lenguaje de programación y demás herramientas utilizadas para su desarrollo.

Para finalizar, se realizará un estudio de las cuestiones relacionadas con la propiedad intelectual de la idea (patentabilidad, protección…), y se analizará si es aplicable o no para este proyecto en concreto.

9.1. Análisis legislativo

El siguiente apartado incluye un listado sobre la legislación vigente en materia, centrándose, sobretodo, en la situación actual en España, marcadas por las leyes del BOE [12]. Cabe destacar que, en el campo de la robótica asistencial, las leyes no están formalizadas ya que es una tecnología relativamente temprana, por lo que recurriremos a las leyes aplicables para nuestro proyecto en otros ámbitos:

Leyes de propiedad intelectual (actualizado a 5 de julio de 2017)

- Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia.

- Real Decreto de 3 de septiembre de 1880 por el que se aprueba el Reglamento para la ejecución de la Ley de 10 de enero de 1879 sobre propiedad intelectual. [Inclusión parcial].

- Real Decreto 635/2015, de 10 de julio, por el que se regula el depósito legal de las publicaciones en línea.

- Real Decreto 281/2003, de 7 de marzo, por el que se aprueba el Reglamento del Registro General de la Propiedad Intelectual.

Legislación sobre riesgos laborales y seguridad (actualizado a 9 de junio de 2017)

- Ley 31/1995, de 8 de noviembre, de prevención de Riesgos Laborales.
- Real Decreto 486/1997, de 14 de abril, por el que se establecen las disposiciones mínimas de seguridad y salud en los lugares de trabajo.
- Real Decreto 488/1997, de 14 de abril, sobre disposiciones mínimas de seguridad y salud relativas al trabajo con equipos que incluyen pantallas de visualización.
- Real Decreto 614/2001, de 8 de junio, sobre disposiciones mínimas para la protección de la salud y seguridad de los trabajadores frente al riesgo eléctrico.

Ayudas a la comunidad científica para nuevas investigaciones

- BOE-A-2015-5581, Resolución de 14 de mayo de 2015, de la Secretaría de Estado de Telecomunicaciones y para la Sociedad de la Información, por la que se convocan ayudas para la realización de proyectos en el marco de la Acción Estratégica de Economía y Sociedad Digital, dentro del Plan de Investigación Científica y Técnica y de Innovación 2013-2016.
- BOE-A-2017-9215, Orden EIC/742/2017, de 28 de julio, por la que se establecen las bases para la concesión de apoyo financiero a proyectos de I+D+I en el ámbito de la industria conectada 4.0.

Avances en legislación a nivel europeo (actualizado a febrero de 2017)

El pasado 16 de febrero de 2017, el proceso hacia una regulación europea de los aspectos civiles de la robótica y de la Inteligencia Artificial dio un importante paso adelante con el acuerdo del Parlamento Europeo de solicitar a la Comisión una propuesta de Directiva sobre la materia, incluyendo una serie de recomendaciones y contenidos detallados con una fuerte apuesta por la ética [13]. Los principales contenidos del documento se indican a continuación:

- Impacto económico de la robótica e inteligencia artificial, requiriendo un enfoque europeo y mayor apoyo económico a emprendedores como a Estados Miembros y a la Unión Europea.
- Análisis de los riesgos, valorando la seguridad humana y la salud, incluyendo la posible conexión emocional humano-robot.

- Propuesta de creación de una Agencia Europea para robótica e inteligencia artificial, dotada económicamente y con personal experto en la materia.

- Necesidad de una regulación (actualmente inexistente) de derechos de propiedad intelectual y flujo de datos, en coherencia con el Reglamento Europeo de Protección de datos y sus previsiones de privacidad por diseño y por defecto. Por ejemplo, a la hora de instalar cámaras y sensores en los robots.

- Conveniencia de la estandarización, coordinada con las normas ESO e ISO, facilitando los tests de prototipos en entornos reales, por ejemplo de medios de transporte autónomos (vehículos, drones …).

- Especial atención a los problemas éticos pero a la vez a las ventajas del desarrollo de robots de cuidado de personas mayores o de los robots médicos.

- En materia de educación y empleo, destaca la llamada a que las mujeres jóvenes desarrollen sus carreras en el entorno digital.

- Impacto ambiental.

- La responsabilidad civil se considera un aspecto crucial, sugiriéndose un seguro obligatorio y un Registro para los robots, por categorías.

9.2. Estándares técnicos

La estandarización o normalización es el proceso de elaborar, aplicar y mejorar las normas que se aplican a distintas actividades científicas, industriales o económicas, con el fin de ordenarlas y mejorarlas [14].

En este apartado se incluirán los estándares técnicos referentes a la robótica y a la programación en C++, ya que son los principales recursos empleados para el desarrollo del proyecto.

Los módulos de procesado y control del robot se han desarrollado en C++, siguiendo el estándar C++14, publicado en 2014 como norma **ISO/IEC 14882:2014**. La versión más reciente del estándar de este lenguaje es la C++17. Sin embargo, la especificación final de C++17 fue aprobada por el comité de estandarización de C++ (WG21) de la Organización Internacional de Normalización (ISO) el 6 de septiembre de 2017 y está pendiente de la publicación oficial definitiva por parte de ISO.1.

Respecto a la comunicación con el robot asistencial AMOR, realizada mediante bus CAN (Controller Area Network), está estandarizada en la norma ISO 11898.

9.3. Propiedad intelectual

Los derechos de propiedad intelectual son similares a cualquier otro derecho de propiedad: permiten al creador o titular de una patente, marca o derecho de autor, gozar de los beneficios que derivan de su obra o de la inversión realizada en relación con una creación. Esos derechos están consagrados en el Artículo 27 de la Declaración Universal de Derechos Humanos, que contempla el derecho a beneficiarse de la protección de los intereses morales y materiales resultantes de la autoría de las producciones científicas, literarias o artísticas.

Una patente es un derecho exclusivo concedido sobre una invención – el producto o proceso que constituye una nueva manera de hacer algo, o propone una nueva solución técnica a un problema. El titular de una patente goza de protección para su invención; la protección se concede durante un período limitado, que suele ser de 20 años [14].

Si analizamos la situación actual en el campo de la robótica, podemos diferenciar dos modelos de actuación:

- En la fase previa a la comercialización, la innovación se basa fundamentalmente en plataformas abiertas colaborativas, como el Sistema Operativo para Robots (ROS). Estas plataformas invitan a terceros a usar o mejorar el contenido existente mediante acuerdos de licencia abierta (por ejemplo, Creative Commons, la Licencia Pública General de la GNU o una licencia de software libre), agilizando la experimentación y el desarrollo de prototipos. Las plataformas colaborativas permiten a sus usuarios compartir la considerable inversión inicial, evitar la duplicación de esfuerzos y perfeccionar los enfoques existentes. Dichas plataformas existen tanto para el desarrollo de programas como de equipos informáticos.

- Sin embargo, cuando las empresas innovadoras invierten en sus propias iniciativas de I+D, tienden a proteger más sus invenciones, especialmente cuando éstas se usan para diferenciar el producto final, entrando en juego la publicación de patentes.

Por otro lado, la complejidad tecnológica de los sistemas robóticos ha dado lugar a que, aquellas empresas que quieren proteger sus innovaciones, opten por una solución alternativa: los secretos comerciales.

En el pasado, aquellas empresas que deseaban proteger sus avances tecnológicos mediante patente (sobre todo aquellos a décadas de ser usados en

productos comercializables), gastaron grandes cantidades de dinero en patentes pero apenas obtuvieron beneficios, ya que llegado el momento de la comercialización las patentes ya habían vencido. Por tanto, la solución del secreto comercial en este ámbito es una de las mejores opciones, permitiendo además el ahorro en costes de presentación de solicitudes de patente.

Junto con las patentes, los diseños industriales que protegen la apariencia de un robot (su forma y figura) también desempeñan un papel importante, al mejorar la capacidad de comercialización de los productos y ayudar a las empresas a garantizar el rendimiento de sus inversiones en I+D [15].

Finalmente, si se aplica el anterior análisis para nuestro proyecto, nos encontramos en una fase previa a la comercialización, donde se están generando pequeñas aportaciones para la construcción de un sistema completo de robot asistencial operativo, y que requiere aún de numerosas mejoras y aportaciones por parte de la comunidad de desarrolladores. Es por esto que, por el momento, no se contemple la posibilidad de patentar el resultado, sino que se haya incluido en una plataforma online, Github, para la contribución de otros programadores en el desarrollo de nuevas versiones.

10. Entorno socio-económico

En este apartado se mostrará un detalle del presupuesto de los materiales y mano de obra empleados para el desarrollo de este proyecto.

Dentro de los costes materiales incluimos el coste de elementos que se han usado durante el desarrollo del TFG aunque no se hayan comprado explícitamente para este, como sucede con el robot asistencial AMOR, o el ordenador portátil con el que se ha trabajado.

Para finalizar, se incluirá un análisis del impacto socio-económico del proyecto, teniendo en cuenta tanto aspectos sociales, económicos, éticos como ambientales.

10.1. Presupuesto

CÓDIGO	RESUMEN	CANTIDAD	PRECIO	IMPORTE
	APARTADO 01 ADQUISICIÓN DEL ROBOT Y ACCESORIOS			
01.01	**UD. MANIPULADOR DE SERVICIO AMOR 7 DOF** Compra del robot manipulador asistencial AMOR 7 DOF con todos sus accesorios básicos. Se incluye CD con la API de AMOR y ejemplos de uso. Unidad completa, totalmente instalada, probada y en perfecto estado de funcionamiento.	1	23.462	23.462
01.02	**UD. PINZA MODULAR** Compra de la garra modular de AMOR con almohadillas anti-deslizamiento y resistentes al agua (a más de 60º). Fuerza máxima de la pinza de hasta 35 N. Máxima apertura entre las pinzas 90 mm. Potencia: 20V, 0.2 A. Interfaz: CAN	1	500	500
01.03	**UD. MARCO DE ANCLAJE PARA SENSORES** Compra del marco universal de anclaje para sensores de AMOR. Diseñado especialmente para el extremo del brazo robótico de AMOR. Puede emplearse para unir una cámara a la muñeca del robot.	1	30	30

01.04	**UD. FUENTE DE ALIMENTACIÓN PARA LABORATORIO** Compra de la fuente de alimentación de AMOR. Dimensiones (w x h x d): 110 x 58 x 226 mm. Potencia 150W. Conexiones: buses de 4mm. Salidas: 1, Voltaje de salida: 22-29 V. Corriente de salida: 5,2 A. Entradas: IEC C14. Voltaje de entrada: 90-264V/AC. Peso: 1,65kg.	1	300	300
01.05	**UD. TRÍPODE** Compra del trípode sustentador de AMOR. Peso: 4kg. Dimensiones (w x h x d): 580 x 450 x 580 mm.	1	50	50
01.06	**ESD CAN-USB/2 Interface** Compra del adaptador USB-CAN para USB 2.0. Interfaz de alta velociad con tasa de transferencia de datos de 480 Mbit/s. Interfaz CAN diferencial con aislamiento eléctrico, 1 Mbit/s, ISO 11898. Alimentación por USB. Con carcasa de aluminio de dimensiones 55mm x 25 mm. Microcontrolador ARM LPC2292.	1	50	50
	TOTAL APARTADO 01 ADQUISICIÓN DEL ROBOT Y ACCESORIOS			24.392
	APARTADO 02 SISTEMAS DE MEDICIÓN Y CONTROL			
02.01	**UD. SENSORES SI1143 PARALLAX** Compra de sensores infrarrojos SI1143 de Parallax con fotodiodos y un circuito conductor para tres leds. Voltaje: 3,3 V. Peso: 14 g. Dimensiones: 3 x 3 x 1 in. Modelo circular. Interfaz I2C (maestro-esclavo) mediante los pines SLC y SDA. El código para su programación requiere JeeLib.	43	14	602
02.02	**UD. PLACA ARDUINO MEGA ADK** Arduino MEGA ADK basada en microcontrolador ATmega2560. Dispone de 54 pines I/O, 16 entradas analógicas, 4 puertos UARTs, cristal de 16 MHz, conector USB, conector alimentación Jack, pines para ICSP y botón de RESET. Alimentación: 5V Entrada: 7-12V. Límites (max): 5,5-16V. Corriente por pin: 40mA. Corriente sobre pin 3,3V:50mA. Reloj: 16 MHz.	3	39,51	118,53

02.03	**UD. JOYSTICK 3D CONNEXION SPACENAVIGATOR** Compra del joysctick Connexion SpaceNavigaror. Sensor de seis grados de movimientos (6DoF) de 3Dconnexion. Dimensiones (L x An x Al): 78 mm x 78 mm x 53 mm. Peso: 479 g. El paquete contiene CD ROM con controladores, demostraciones y herramientas.	1	132	132
02.04	**UD. PC TOSHIBA SATELLITE i5** Adquisición del ordenador portátil Toshiba Satellite i5. Modelo: P755-3DV20. Procesador: Core® i5 2da. Generación. Familia: Satellite®. Tamaño de la Pantalla: 15.6". Tipo de Pantalla: LED 3D . Tamaño de la Memoria Ram: 6 Gb. Capacidad Disco Duro: 750 Gb. Unidad óptica: Dvd±Rw Blu-ray.	1	829	829
	TOTAL APARTADO 02 INSTALACIÓN DE SISTEMAS DE MEDICIÓN Y CONTROL			1681,53
	APARTADO 03 MANO DE OBRA Horas dedicadas por el ingeniero para la realización del proyecto	360	30	10800
	TOTAL APARTADO 03 MANO DE OBRA			10800
	TOTAL PRESUPUESTO DEL PROYECTO			36.873,53

10.2. Impacto socio-económico

En relación al impacto socio-económico que existe en el ámbito de la robótica, se pueden distinguir dos puntos de vista totalmente diferentes. Si analizamos el impacto que generan los robots en la sociedad según su inclusión en el mercado laboral, se puede observar cierta preocupación y rechazo al ver peligrar sus puestos de trabajo. Por otro lado, el uso de robots en áreas como la medicina tiene una acogida más positiva, al ser capaces de prestar asistencia a personas de forma ininterrumpida.

En relación al impacto económico y social de la inclusión de los robots en el mercado laboral, es un tema que siempre ha estado en el punto de mira, ya que la sociedad lo cataloga como una amenaza de cara al mundo laboral.

Hace unos años se pensaba que los principales puestos de trabajo que podían ser afectados por la automatización, serían aquellos de baja cualificación. Sin embargo, en la actualidad los robots son capaces de sustituir también empleos de conocimientos intermedios como la sanidad, el transporte o tareas administrativas, ya que contienen numerosas tareas repetitivas y fáciles de automatizar [16].

Según una publicación del profesor Salvador del Rey, catedrático de Derecho, estima que el coste de crear un robot caerá un 20% el próximo año, al tiempo que su rendimiento aumentará un 5%. Además, mientras que las personas doblan su productividad cada 10 años, los robots, como mínimo, lo hacen cada cuatro. Sin olvidar que el tiempo de amortización de un robot era en 2015 de 5,3 años y en 2025 se reducirá a 1,3 años.

En la siguiente imagen se puede observar una gráfica con el porcentaje de trabajadores por país cuyo puesto de trabajo se encuentra en riesgo de ser sustituido por un autómata [17].

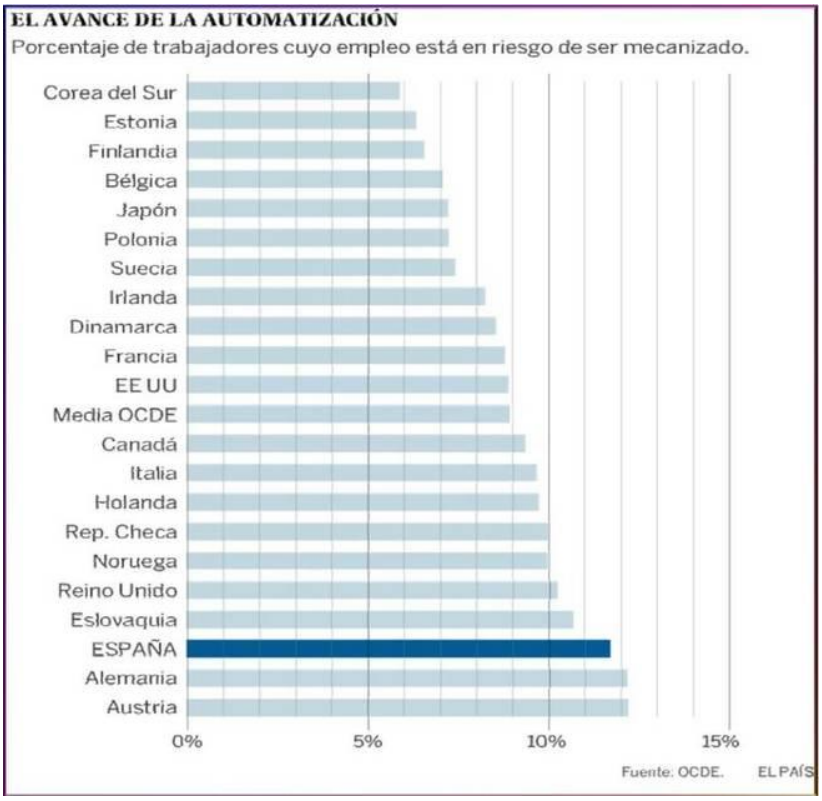

Figura 42. Paises con alto riesgo de sustitución de empleados por autómatas

En relación al impacto de la robótica en la medicina, los robots se están abriendo paso con éxito como asistentes terapéuticos. Múltiples experiencias e investigaciones muestran que resultan de utilidad para mejorar la calidad de vida de niños y mayores en hospitales y residencias de ancianos, para estimular el aprendizaje, para desarrollar las habilidades sociales en personas con trastornos del espectro autista o para potenciar y facilitar la terapia de personas con alguna discapacidad mental o física.

Para finalizar el análisis, haremos una breve mención al impacto ambiental, aunque, al tratarse de un desarrollo digital, apenas cobra importancia. Únicamente se podría destacar el elevado consumo de electricidad, tanto a la hora de realizar las investigaciones como durante la utilización del robot por el usuario final, considerándolo como un impacto negativo hacia el medio ambiente.

11. Cronograma

AÑO 2017	ENE				FEB				MAR				ABR				MAY				JUN				JUL				AGO				SEPT			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1. Investigación del entorno del proyecto																																				
Estudio proyecto previo con adquisición datos sensores (Arduino)																																				
Recopilación documentación para el proyecto actual																																				
Repaso conocimientos C++																																				
Familiarización con API AMOR																																				
2. Toma de medidas																																				
Cálculo del rango de medida de los sensores																																				
Pruebas de detección con distintos tipos de materiales																																				
3. Procesado de los datos																																				
Análisis resultados y desarrollo código de procesado de señal																																				
4. Desarrollo programa con trayectoria programada																																				
Establecer trayectoria programada en robot																																				
Configurar lectura sensores y niveles de alerta																																				
Configurar control del robot (actuador)																																				
5. Desarrollo módulo sensores																																				
Implementación programa sensores																																				
Integración con Space Navigator																																				
6. Pruebas y corrección de errores																																				
Pruebas módulo sensores																																				
Visualización del módulo sensores integrado en sistema completo																																				
Grabación de vídeos para la presentación																																				
7. Redacción memoria TFG																																				
Recopilación información y resultados obtenidos																																				
Redacción del documento																																				
Revisión del documento																																				
Entrega del proyecto																																				

12. Bibliografía

[1] OMS. (2014). Estadísticas sanitarias mundiales.

[2] Núñez, P. V. (2014). Robot socialmente asistencial y de interacción multimodal autónoma. Inge@UAN, 15.24.

[3] "Safely among us", IEEE-RAS Magazine, Volume 11, Issue 2, Special Issue on Dependability of Human-Friendly Robots, June 2004.

[4] Martin F. Stoelen, V. F. (2013). Adaptive Collision-Limitation Behavior for an Assistive Manipulator.

[5] Device, M. (s.f.). Modern Device. Recuperado el Junio de 2017, de https://moderndevice.com/product/si1143-proximity-sensors/

[6] Arduino.cc. (s.f.). Recuperado el 05 de 2017, de https://www.arduino.cc/en/Main/ArduinoBoardMegaADK

[7] Arduino.cc SoftwareSerial. (s.f.). Recuperado el 05 de 2017, de https://www.arduino.cc/en/Reference/SoftwareSerial

[9] IBM. (s.f.). IBM. Recuperado el 09 de 2017, de Modelo cliente/servidor: https://www.ibm.com/support/knowledgecenter/es/SSAL2T_8.2.0/com.ibm.cics.tx.doc/concepts/c_clnt_sevr_model.html

[10] AMOR API library. (s.f.). Recuperado el 07 de 2017, de http://www.amorrobot.com/apidoc/index.html

[11] Wikipedia (Complemento o plug in). (s.f.). Recuperado el 09 de 2017, de https://es.wikipedia.org/wiki/Complemento_(inform%C3%A1tica)

[12] Agencia Estatal Boletin Oficial del Estado. (s.f.). Recuperado el 09 de 2017, de https://www.boe.es/diario_boe/

[13] Fernández, C. B. (17 de 02 de 2017). El Pleno del Parlamento Europeo aprueba la propuesta de regulación sobre robótica e inteligencia artificial. Diario La Ley.

[14] Keisner, A. (2016). Tecnologías revolucionarias: robótica y P.I. OMPI Revista.

[15] Keisner, A. (2017). ¿Qué es la propiedad intelectual? Organización Mundial de la Propiedad Intelectual.

[16] Rius, M. (21 de 11 de 2016). Asistidos por autómatas. La Vanguardia.

[17] Sánchez, C. (15 de 08 de 2016). El uso de robots se acelera y amenaza con destruir decenas de miles de empleos. El Confidencial, págs. https://www.elconfidencial.com/economia/2016-08-15/robots-automatizacion-ccoo-empleo-maquinas-industria_1245720/.

[18] G. Hirzinger, A. Albu-Schaeffer, M. Hahnle, I. Schaefer, and N. Sporer, “On a new generation of torque controlled light-weight robots”, 2001 IEEE International Conference of Robotics and Automation, Seoul, K, 2001.

[19]Wikipedia (Normalización). (s.f.). Recuperado el 09 de 2017, de https://es.wikipedia.org/wiki/Normalizaci%C3%B3n

[20]YARP. (s.f.). Recuperado el 5 de 2017, de www.yarp.it

[21] AmorRobot. (s.f.). Recuperado el 9 de 2017, de www.amorrobot.com

[22] K. Severinson-Eklundh, A. Green, and H. Huettenrauch, “Social and collaborative aspects of interaction with a service robot”, Workshop on “Robot as Partner: an exploration of social robot”, IEEE-RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland, 2002

[24] A. Albu-Schäffer, C. Ott, and G. Hirzinger, “A unified passivity based control framework for position, torque and impedance control of flexible joint robots”, Int. Symposium on Robotics Research 2005, San Francisco, CA, USA.

[25] Streaming Cartesian Control. Recuperado el 9 de 2017 de www.github.com https://github.com/roboticslab-uc3m/kinematics-dynamics/blob/streaming-cartesian-control/programs/streamingDeviceController/StreamingDeviceController.cpp

[26] Sensors. Recuperado el 9 de 2017 de www.github.com https://github.com/roboticslab-uc3m/amor-main/tree/develop/share/sensors/scripts

[27] Proximity Sensors Client. Recuperado el 9 de 2017 de www.github.com https://github.com/roboticslab-uc3m/yarp-devices/tree/1282e1511b5fda2db7deb3ca0a93c6b99427fbf1/libraries/BodyYarp/ProximitySensorsClient

13. Anexos

Anexo 1. Sensores_AMOR.ino

```
#include <SI114.h>

const int portForSI114 = 0;        // change to the JeeNode port
number used

/*
 For Arduino users just use the following pins for various port
settings
 Or use port 0 for traditional SDA (A4) and SCL (A5)
 Connect pins with 10k resistors in series

 JeeNode users just set the appropriate port

 JeeNode Port  SCL ('duino pin)  SDA ('duino pin)
 0             19 (A5)       18 (A4)
 1             4             14 (A0)
 2             5             15 (A1)
 3             6             16 (A2)
 4             7             17 (A3)
 */
//#define ALISADO
#define RESOLUCION 3
#define SEND
#define alfa 0.8
#define Nsensor 16
#define DIRBASE 0x10
#define PINBASE 2
#define asignarPin(n)  ((n)+PINBASE<18 ?(n)+PINBASE :-1)
#define desplazar 4*(4-RESOLUCION)
enum{
  tred=0,tIR1,tIR2};

unsigned int sensor[Nsensor][3];

unsigned long red;        // read value from visible red LED
unsigned long IR1;        // read value from infrared LED1
unsigned long IR2;        // read value from infrared LED2

boolean entradasUtilizadas[16]={1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
//Poner un 1 en su correspondiente lugar si se utiliza ese pin o un 0
si se deja al aire, siendo el 1° el pin 2
                                                                  //Y
el ultimo el A3.  D2,D3,D4,D5,D6,D7,D8,D9
D10,D11,D12,D13,A0,A1,A2,A3.

PortI2C myBus (portForSI114);
PulsePlug pulse (myBus);

void setup(){
  Serial.begin(115200);
  Serial.println("\n INICIALIZACION");
  for(uint8_t i=0;i<Nsensor;i++){
```

```
   if (entradasUtilizadas[i]==1){

    pulse.setAddress(0x5A);
    digitalWrite(asignarPin(i),LOW);
    digitalWrite(asignarPin(i),HIGH);
    Serial.print("-------PIN: ");
    Serial.println(asignarPin(i));
    delay(70); //>40ms de encendido de sensor
    pulse.setAddress(0x5A);
    if (pulse.isPresent()) {
      Serial.print("OK -> SI114x Pulse Sensor found on Port ");
      Serial.println(portForSI114);
    }
    else{
      Serial.print("No SI114x found on Port ");
      Serial.println(portForSI114);
    }

    initPulseSensor(i);
  }
  else {
      Serial.print("-----PIN: ");
      Serial.println(asignarPin(i));
      Serial.println(" No utilizado");
    }
    }
}
void loop(){
  readPulseSensor();
  #ifdef SEND
  if(Serial.available()>0){
    char d=Serial.read();
    if(d=='X'){
      Serial.print('X');//Cambiar millis() por 'X'

      for(uint8_t j=0;j<Nsensor;j++){
        if((sensor[j][0]>>desplazar)<0x100){
          Serial.print("0");
        }
        if((sensor[j][0]>>desplazar)<0x10){
          Serial.print("0");
        }

        Serial.print(sensor[j][0]>>desplazar,HEX);
      }
      Serial.println();
      Serial.print('Y');
      for(uint8_t j=0;j<Nsensor;j++){
        if((sensor[j][1]>>desplazar)<0x100){
          Serial.print("0");
        }
        if((sensor[j][1]>>desplazar)<0x10){
          Serial.print("0");
        }

        Serial.print(sensor[j][1]>>desplazar,HEX);
      }
      Serial.println();
      Serial.print('Z');
      for(uint8_t j=0;j<Nsensor;j++){
        if((sensor[j][2]>>desplazar)<0x100){
```

```
          Serial.print("0");
        }
        if((sensor[j][2]>>desplazar)<0x10){
          Serial.print("0");
        }

        Serial.print(sensor[j][2]>>desplazar,HEX);
      }
      Serial.println();
    }
    //Serial.flush();
  }
}

void initPulseSensor(uint8_t i){
  pulse.setAddress(0x5A);    //direccion de defecto de los sensores
  Serial.print("-----SENSOR ");
  Serial.println(i+1);

  pulse.setReg(PulsePlug::HW_KEY, 0x17);
  // pulse.setReg(PulsePlug::COMMAND, PulsePlug::RESET_Cmd);

  Serial.print("PART: ");
  Serial.print(pulse.getReg(PulsePlug::PART_ID));
  Serial.print(" REV: ");
  Serial.print(pulse.getReg(PulsePlug::REV_ID));
  Serial.print(" SEQ: ");
  Serial.println(pulse.getReg(PulsePlug::SEQ_ID));

  pulse.setReg(PulsePlug::INT_CFG, 0x03);       // turn on interrupts
  pulse.setReg(PulsePlug::IRQ_ENABLE, 0x10);    // turn on interrupt
on PS3
  pulse.setReg(PulsePlug::IRQ_MODE2, 0x01);     // interrupt on ps3
measurement
  pulse.setReg(PulsePlug::MEAS_RATE, 0x84);     // see datasheet
  pulse.setReg(PulsePlug::ALS_RATE, 0x08);      // see datasheet
  pulse.setReg(PulsePlug::PS_RATE, 0x08);       // see datasheet

  // Current setting for LEDs pulsed while taking readings
  // PS_LED21  Setting for LEDs 1 & 2. LED 2 is high nibble
  // each LED has 16 possible (0-F in hex) possible settings
  // read the
  pulse.setReg(PulsePlug::PS_LED21, 0x38);      // LED current for 2
(IR1 - high nibble) & LEDs 1 (red - low nibble)
  pulse.setReg(PulsePlug::PS_LED3, 0x02);       // LED current for LED
3 (IR2)

  Serial.print( "PS_LED21 = ");
  Serial.println(pulse.getReg(PulsePlug::PS_LED21), BIN);
  Serial.print("CHLIST = ");
  Serial.println(pulse.readParam(0x01), BIN);

  pulse.writeParam(PulsePlug::PARAM_CH_LIST, 0x77);         // all
measurements on

  // increasing PARAM_PS_ADC_GAIN will increase the LED on time and
ADC window
  // you will see increase in brightness of visible LED's, ADC output,
& noise
  // datasheet warns not to go beyond 4 because chip or LEDs may be
damaged
```

```
  pulse.writeParam(PulsePlug::PARAM_PS_ADC_GAIN, 0x00);


  // You can select which LEDs are energized for each reading.
  // The settings below turn on only the LED that "normally" would be
read
  // ie LED1 is pulsed and read first, then LED2 is pulsed and read
etc.
  pulse.writeParam(PulsePlug::PARAM_PSLED12_SELECT, 0x21);  // 21 =
LED 2 & LED 1 (red) resp.
  pulse.writeParam(PulsePlug::PARAM_PSLED3_SELECT, 0x04);   // 4 = LED
3 only

  // Sensors for reading the three LEDs
  // 0x03: Large IR Photodiode
  // 0x02: Visible Photodiode - cannot be read with LEDs on - just for
ambient measurement
  // 0x00: Small IR Photodiode
  pulse.writeParam(PulsePlug::PARAM_PS1_ADCMUX, 0x03);      // PS1
photodiode select
  pulse.writeParam(PulsePlug::PARAM_PS2_ADCMUX, 0x03);      // PS2
photodiode select
  pulse.writeParam(PulsePlug::PARAM_PS3_ADCMUX, 0x03);      // PS3
photodiode select

  pulse.writeParam(PulsePlug::PARAM_PS_ADC_COUNTER, B01110000);    //
B01110000 is default
  pulse.setReg(PulsePlug::COMMAND, PulsePlug::PSALS_AUTO_Cmd);     //
starts an autonomous read loop
  // Serial.println(pulse.getReg(PulsePlug::CHIP_STAT), HEX);

  Serial.println("-Cambio de addr-");
  Serial.print("DIR PULSE default: ");
  Serial.println(pulse.getAddress(),HEX);
  pulse.setReg(PulsePlug::COMMAND, PulsePlug::NOP_cmd);
  Serial.print("Dir VIEJA sensor: ");
  Serial.println(pulse.readParam(PulsePlug::PARAM_I2C_ADDR), HEX);

  pulse.writeParam(PulsePlug::PARAM_I2C_ADDR,byte(DIRBASE+i));
// Cambiar direccion
  pulse.setReg(PulsePlug::COMMAND, PulsePlug::BUSADDR_cmd);
  pulse.setAddress(DIRBASE+i);
  Serial.print("DIR PULSE: ");
  Serial.println(pulse.getAddress(),HEX);
  Serial.print("Dir NUEVA sensor: ");
  Serial.println(pulse.readParam(PulsePlug::PARAM_I2C_ADDR), HEX);
  delay(10);//espera a que el cambio se haga efectivo
  if (pulse.isPresent()) {
    Serial.print("OK -> SI114x Pulse Sensor found on Port ");
    Serial.println(portForSI114);
  }
  else{
    Serial.print("No SI114x found on Port ");
    Serial.println(portForSI114);
    delay(10);
  }
  Serial.println("end init");
}

void readPulseSensor(){
```

```
  for(uint8_t n=0;n<Nsensor;n++){
   if (entradasUtilizadas[n]==1){
    int signalSize;
    red = 0;
    IR1 = 0;
    IR2 = 0;



    pulse.setAddress(DIRBASE+n);

    pulse.fetchData();
    //delay(5);

    red += pulse.ps1;
    IR1 += pulse.ps2;
    IR2 += pulse.ps3;


    // para usar alisado exponencial
#ifdef ALISADO
    red = alisado(red,n,tred);
    IR1 = alisado(IR1,n,tIR1);
    IR2 = alisado(IR2,n,tIR2);
#endif
    sensor[n][0]=red;
    sensor[n][1]=IR1;
    sensor[n][2]=IR2;
  }
 else {
       sensor[n][0]=0xFFFF;
      sensor[n][1]=0xFFFF;
      sensor[n][2]=0xFFFF;
     }
    }



#endif
}

#ifdef ALISADO
unsigned long alisado(unsigned long val, uint8_t sensor, uint8_t
tipo){
  static uint32_t valmedia[Nsensor][3];  //F-1
  //static unsigned long valantes[Nsensor][3];  //A-1
  unsigned long valfin;
  static boolean firsttime= 1;
  //long valsigned;
  if (firsttime){ //Inicialización de la matriz
    firsttime= 0;
    for(uint8_t i=0;i<Nsensor;i++){
      for(uint8_t j=0;j<3;j++){
        valmedia[i][j]=0;
      }
    }
  }
  if(val==65535)  //Ignora los fallos en la comunicacion
    val=valmedia[sensor][tipo];
  //valsigend=val-32767;
```

```
  //F = F-1 + ALFA*(A-1 - F-1)
  valfin = (uint32_t)(valmedia[sensor][tipo]+(signed
long)(float(alfa)*(signed long)(val-valmedia[sensor][tipo])));
  // valantes[sensor][tipo]=val;
  valmedia[sensor][tipo]=(uint32_t)valfin;
  return valfin;
}
#endif
```

Anexo 2. Comunicación_AMOR_MEGA.ino

```
#include <TimerThree.h>

#include <SoftwareSerial256.h>;//Libreria modificada

#define size_w 5
#define MAXBUFFER 256
#define Nsensores1 16
#define Nsensores2 16
#define Nsensores3 16
#define LedRojo 6
#define LedVerde 5



char datos[MAXBUFFER];

char d;
uint8_t count=0, sensor=0, IR=0;
boolean flag1=HIGH;
boolean flag2=HIGH;
boolean flag3=HIGH;
boolean flag4=HIGH;

//SoftwareSerial Serial3(10,11);
void refresh(){
  Serial1.flush();
  Serial2.flush();
  Serial3.flush();
  flag1=1;
  flag2=1;
  flag3=1;
  flag4=1;
}
void setup(){
  Serial1.begin(115200);
  Serial2.begin(115200);
  Serial3.begin(115200);
  Serial.begin(115200);
  //Serial3.begin(115200);
  pinMode(LedRojo,OUTPUT);
  pinMode(LedVerde,OUTPUT);
  Serial.println("INICIALIZANDO");
  parpadeoRojo(7);  //Espera a la inicialización de los sensores en
los otros arduinos mientras parpadea el led rojo 7 segundos
  Serial.println("OK");
  digitalWrite(LedVerde,HIGH);
  Timer3.initialize(77500); //en microsegundos
  Timer3.attachInterrupt(refresh);
}

void loop(){
  //Serial1.flush();
  //while(Serial3.available()>0){};
  //delay(1);
  //Serial.println(millis());
  if(flag1){
    Serial1.print('X');
```

```
    //delay(10);
    flag1=0;
  }
  if(Serial1.available()>=(Nsensores1*3+2)*3){
    //Serial.print(millis());
    Serial.println('I');
    serial1GetString(datos,MAXBUFFER);
//    Serial.println(millis());
    Serial.println((char*)datos);
    //flag1=1;
  }

  if(flag2){
    Serial2.print('X');
    //delay(10);
    flag2=0;
  }
  if(Serial2.available()>=(Nsensores2*3+2)*3){
    Serial.println('J');
    serial2GetString(datos,MAXBUFFER);
//    Serial.println(millis());
    Serial.println((char*)datos);
    //flag2=1;
  }



  //Serial3.listen();
  //Serial3.flush();
  //while(Serial3.available()>0){};
  //delay(1);
  if(flag3){
    Serial3.print('X');
    delay(10);
    flag3=0;
  }
  if(Serial3.available()>=(Nsensores3*3+2)*3){
    Serial.println('K');
    Serial3GetString(datos,MAXBUFFER);
//    Serial.println(millis());
    Serial.println((char*)datos);
    //flag3=1;
  }
}
int serial1GetString(char *string, int max)
{
  unsigned i=0;
  char sIn;
  unsigned long m;
  // Queremos que la cadena se rellene hasta max-2 para que en el
carácter
  // max-1 (el último) podamos meter el terminador \0
  --max;
  while (Serial1.available()>0 && i<max){
    sIn=Serial1.read();
    string[i++]=sIn;

    //delayMicroseconds(10);
  }
  string[i++]='\0';
  return i;
```

```
}
int serial2GetString(char *string, int max)
{
  unsigned i=0;
  char sIn;
  unsigned long m;
  // Queremos que la cadena se rellene hasta max-2 para que en el
carácter
  // max-1 (el último) podamos meter el terminador \0
  --max;
  while (Serial2.available()>0 && i<max){
    sIn=Serial2.read();
    string[i++]=sIn;

    //delayMicroseconds(10);
  }
  string[i++]='\0';
  return i;
}
int Serial3GetString(char *string, int max)
{
  //Serial3.listen();
  unsigned i=0;
  char sIn;
  unsigned long m;
  // Queremos que la cadena se rellene hasta max-2 para que en el
carácter
  // max-1 (el último) podamos meter el terminador \0
  --max;
  while (Serial3.available()>0 && i<max){
    sIn=Serial3.read();
    string[i++]=sIn;
    //if(Serial3.overflow())
    //  Serial.println("OVERFLOW");
    //delayMicroseconds(10);
  }
  string[i++]='\0';
  return i;
}

void parpadeoRojo(int repeticiones){
  for (int i=0;i<repeticiones;i++){
    digitalWrite(LedRojo,HIGH);
    delay(500);
    digitalWrite(LedRojo,LOW);
    delay(500);
  }
}


  //************************AÑADIDOS EN ESTA
VERSION*************************
  //Funcion parpadeo con los leds. Modificados el numero de bits que
recibe como parametro el Serial Avaible
```

Anexo 3. SerialStream.h

```
/*
 * Author: Terraneo Federico
 * Distributed under the Boost Software License, Version 1.0.
 */

#include <string>
#include <stdexcept>
#include <boost/system/error_code.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/iostreams/stream.hpp>
#include <boost/iostreams/categories.hpp>
#include <boost/date_time/posix_time/posix_time_duration.hpp>

#ifndef SERIALSTREAM_H
#define    SERIALSTREAM_H

/**
 * Thrown if timeout occurs
 */
class TimeoutException: public std::ios_base::failure
{
public:
    TimeoutException(const std::string& arg): failure(arg) {}
};

/**
 * This class contains all the options for a serial port.
 */
class SerialOptions
{
    typedef boost::posix_time::time_duration time_duration;
    typedef boost::posix_time::seconds seconds;

public:
    ///Possible flow controls.
    enum FlowControl { noflow, software, hardware };

    ///Possible parities.
    enum Parity { noparity, odd, even };

    ///Possible stop bits.
    enum StopBits { one, onepointfive, two };

    /**
     * Default constructor.
     */
    SerialOptions() : device(), baudrate(9600), timeout(seconds(0)),
            parity(noparity), csize(8), flow(noflow), stop(one) {}

    /**
     * Constructor.
     * \param device device name (/dev/ttyS0, /dev/ttyUSB0, COM1, ...)
     * \param baudrate baudrate, like 9600, 115200 ...
     * \param timeout timeout when reading, use zero to disable
     * \param parity parity
     * \param csize character size (5,6,7 or 8)
     * \param flow flow control
```

```
     * \param stop stop bits
     *
     */
    SerialOptions(const std::string& device, unsigned int baudrate,
            time_duration timeout=seconds(0), Parity parity=noparity,
            unsigned char csize=8, FlowControl flow=noflow, StopBits
stop=one) :
            device(device), baudrate(baudrate), timeout(timeout),
            parity(parity), csize(csize), flow(flow), stop(stop) {}

    /**
     * Setter and getter for device name
     */
    void setDevice(const std::string& device) { this->device=device; }
    std::string getDevice() const { return this->device; }

    /**
     * Setter and getter for baudrate
     */
    void setBaudrate(unsigned int baudrate) { this->baudrate=baudrate;
}
    unsigned int getBaudrate() const { return this->baudrate; }

    /**
     * Setter and getter for timeout
     */
    void setTimeout(time_duration timeout) { this->timeout=timeout; }
    time_duration getTimeout() const { return this->timeout; }

    /**
     * Setter and getter for parity
     */
    void setParity(Parity parity) { this->parity=parity; }
    Parity getParity() const { return this->parity; }

    /**
     * Setter and getter character size
     */
    void setCsize(unsigned char csize) { this->csize=csize; }
    unsigned char getCsize() const { return this->csize; }

    /**
     * Setter and getter flow control
     */
    void setFlowControl(FlowControl flow) { this->flow=flow; }
    FlowControl getFlowControl() const { return this->flow; }

    /**
     * Setter and getter for stop bits
     */
    void setStopBits(StopBits stop) { this->stop=stop; }
    StopBits getStopBits() const { return this->stop; }

private:
    std::string device;
    unsigned int baudrate;
    time_duration timeout;
    Parity parity;
    unsigned char csize;
    FlowControl flow;
    StopBits stop;
```

```
};

//Forward declaration
class SerialDeviceImpl;

/**
 * \internal
 * Implementation detail of a serial device.
 * User code should use SerialStream
 */
class SerialDevice
{
public:
    typedef char char_type;
    typedef boost::iostreams::bidirectional_device_tag category;

    /**
     * \internal
     * Constructor.
     * \throws ios_base::failure if there are errors with the serial
port.
     * \param options serial port options
     */
    explicit SerialDevice(const SerialOptions& options);

    /**
     * \internal
     * Read from serial port.
     * \throws TimeoutException on timeout, or ios_base::failure if
there are
     * errors with the serial port.
     * Note: TimeoutException derives from ios_base::failure so
catching that
     * allows to catch any exception.
     * Use the clear() member function to go on reading after an
exception was
     * thrown.
     * \param s where to store read characters
     * \param n max number of characters to read
     * \return number of character read
     */
    std::streamsize read(char *s, std::streamsize n);

    /**
     * \internal
     * Write to serial port.
     * \throws ios_base::failure if there are errors with the serial
port.
     * Use the clear() member function to go on reading after an
exception was
     * thrown.
     * \param s
     * \param n
     * \return
     */
    std::streamsize write(const char *s, std::streamsize n);

private:
    /**
     * Callack called either when the read timeout is expired or
canceled.
```

```
     * If called because timeout expired, sets result to
resultTimeoutExpired
     */
    void timeoutExpired(const boost::system::error_code& error);

    /**
     * Callback called either if a read complete or read error occurs
     * If called because of read complete, sets result to
resultSuccess
     * If called because read error, sets result to resultError
     */
    void readCompleted(const boost::system::error_code& error,
            const size_t bytesTransferred);

    boost::shared_ptr<SerialDeviceImpl> pImpl; //Implementation
};

/**
 * SerialStream, an iostream-compatible serial port class.
 * Note: due to a limitation about error reporting with
boost::iostreams,
 * this class *always* throws exceptions on error (timeout, failure,
etc..)
 * so after creating an instance of this class you should alway enable
 * exceptions with the exceptions() member function:
 * \code SerialStream serial; serial.exceptions(ios::failbit |
ios::badbit);
 * \endcode
 * If you don't, functions like getline() will swallow the exceptions,
while
 * operator >> will throw, leading to unconsistent behaviour.
 */
typedef boost::iostreams::stream<SerialDevice> SerialStream;

#endif //SERIALSTREAM_H
```

Anexo 4. SerialStream.cpp

```
/*
 * Author: Terraneo Federico
 * Distributed under the Boost Software License, Version 1.0.
 *
 * v1.01:  Fixed a bug regarding reading after a timeout.
 *
 * v1.00: First release.
 */

#include "serialstream.h"

#include <iostream>
#include <boost/asio.hpp>
#include <boost/bind.hpp>

using namespace std;
using namespace boost;
using namespace boost::asio;

/**
 * Possible outcome of a read. Set by callbacks, read from main code
 */
enum ReadResult
{
    resultInProgress,
    resultSuccess,
    resultError,
    resultTimeout
};

//
// class SerialDeviceImpl
//

/**
 * Contains all data of the SerialDevice class
 */
class SerialDeviceImpl : private boost::noncopyable
{
public:
    /**
     * Construct a SerialDeviceImpl from a SerialOptions class
     * \param options serial port options
     */
    SerialDeviceImpl(const SerialOptions& options);

    io_service io; ///< Io service object
    serial_port port; ///< Serial port object
    deadline_timer timer; ///< Timer for timeout
    posix_time::time_duration timeout; ///< Read/write timeout
    enum ReadResult result;  ///< Used by read with timeout
    streamsize bytesTransferred; ///< Used by async read callback
    char *readBuffer; ///< Used to hold read data
    streamsize readBufferSize; ///< Size of read data buffer
};

SerialDeviceImpl::SerialDeviceImpl(const SerialOptions& options)
        : io(), port(io), timer(io), timeout(options.getTimeout()),
```

```
        result(resultError), bytesTransferred(0), readBuffer(0),
        readBufferSize(0)
{
    try {
        //For this code to work, there should always be a timeout, so
the
        //request for no timeout is translated into a very long
timeout
        if(timeout==posix_time::seconds(0))
timeout=posix_time::hours(100000);

        port.open(options.getDevice());//Port must be open before
setting option


port.set_option(serial_port_base::baud_rate(options.getBaudrate()));

        switch(options.getParity())
        {
            case SerialOptions::odd:
                port.set_option(serial_port_base::parity(
                        serial_port_base::parity::odd));
                break;
            case SerialOptions::even:
                port.set_option(serial_port_base::parity(
                        serial_port_base::parity::even));
                break;
            default:
                port.set_option(serial_port_base::parity(
                        serial_port_base::parity::none));
                break;
        }


port.set_option(serial_port_base::character_size(options.getCsize()));

        switch(options.getFlowControl())
        {
            case SerialOptions::hardware:
                port.set_option(serial_port_base::flow_control(
                        serial_port_base::flow_control::hardware));
                break;
            case SerialOptions::software:
                port.set_option(serial_port_base::flow_control(
                        serial_port_base::flow_control::software));
                break;
            default:
                port.set_option(serial_port_base::flow_control(
                        serial_port_base::flow_control::none));
                break;
        }

        switch(options.getStopBits())
        {
            case SerialOptions::onepointfive:
                port.set_option(serial_port_base::stop_bits(
                        serial_port_base::stop_bits::onepointfive));
                break;
            case SerialOptions::two:
                port.set_option(serial_port_base::stop_bits(
                        serial_port_base::stop_bits::two));
```

```
                break;
            default:
                port.set_option(serial_port_base::stop_bits(
                        serial_port_base::stop_bits::one));
                break;
        }
    } catch(std::exception& e)
    {
        throw ios::failure(e.what());
    }
}

//
// class SerialDevice
//

SerialDevice::SerialDevice(const SerialOptions& options)
                : pImpl(new SerialDeviceImpl(options)) {}

streamsize SerialDevice::read(char *s, streamsize n)
{
    pImpl->result=resultInProgress;
    pImpl->bytesTransferred=0;
    pImpl->readBuffer=s;
    pImpl->readBufferSize=n;

    pImpl->timer.expires_from_now(pImpl->timeout);
    pImpl-
>timer.async_wait(boost::bind(&SerialDevice::timeoutExpired,this,
            boost::asio::placeholders::error));

    pImpl-
>port.async_read_some(buffer(s,n),boost::bind(&SerialDevice::readCompl
eted,

this,boost::asio::placeholders::error,boost::asio::placeholders::bytes
_transferred));

    for(;;)
    {
        pImpl->io.run_one();
        switch(pImpl->result)
        {
            case resultSuccess:
                pImpl->timer.cancel();
                return pImpl->bytesTransferred;
            case resultTimeout:
                pImpl->port.cancel();
                throw(TimeoutException("Timeout expired"));
            case resultError:
                pImpl->port.cancel();
                pImpl->timer.cancel();
                throw(ios_base::failure("Error while reading"));
            default:
            //if resultInProgress remain in the loop
                break;
        }
    }
}

streamsize SerialDevice::write(const char *s, streamsize n)
```

```
{
    try {
        asio::write(pImpl->port,asio::buffer(s,n));
    } catch(std::exception& e)
    {
        throw(ios_base::failure(e.what()));
    }
    return n;
}

void SerialDevice::timeoutExpired(const boost::system::error_code&
error)
{
    if(!error && pImpl->result==resultInProgress) pImpl-
>result=resultTimeout;
}

void SerialDevice::readCompleted(const boost::system::error_code&
error,
        const size_t bytesTransferred)
{
    if(!error)
    {
        pImpl->result=resultSuccess;
        pImpl->bytesTransferred=bytesTransferred;
        return;
    }

    //In case a asynchronous operation is cancelled due to a timeout,
    //each OS seems to have its way to react.
    #ifdef _WIN32
    if(error.value()==995) return; //Windows spits out error 995
    #elif defined(__APPLE__)
    if(error.value()==45)
    {
        //Bug on OS X, it might be necessary to repeat the setup
        //http://osdir.com/ml/lib.boost.asio.user/2008-
08/msg00004.html
        pImpl->port.async_read_some(
                asio::buffer(pImpl->readBuffer,pImpl->readBufferSize),

boost::bind(&SerialDevice::readCompleted,this,boost::asio::placeholder
s::error,
                boost::asio::placeholders::bytes_transferred));
        return;
    }
    #else //Linux
    if(error.value()==125) return; //Linux outputs error 125
    #endif

    pImpl->result=resultError;
}
```

Anexo 5. Amor_sensors_modifier.lua

```
-- loading lua-yarp binding library
require("yarp")

-- Lua 5.2 lacks a global 'unpack' function
local unpack = table.unpack or unpack

local SensorDataProcessor = {}

--
-- Creates a SensorDataProcessor instance.
--
-- @param processor Table of input parameters
--
-- @return SensorDataProcessor
--
SensorDataProcessor.new = function(self, processor)
    local obj = {
        accumulator = "",
        dataReady = false,
        currentSensorData = nil,
        processor = processor,
        stamp = 0
    }

    setmetatable(obj, self)
    self.__index = self
    return obj
end

--
-- Consumes fetched data from sensors.
--
-- @param str String of new data to be appended to the accumulator
--
SensorDataProcessor.accept = function(self, str)
    str = str or ""
    self.accumulator = self.accumulator .. str
    self:tryConsume()
end

--
-- Calls further processing methods if conditions are satisfied,
-- sets flag if data is ready to be forwarded to receiver.
--
SensorDataProcessor.tryConsume = function(self)
    if self.processor.evaluateCondition(self.accumulator) then
        self.currentSensorData, self.accumulator =
self.processor.process(self.accumulator)

        if self.currentSensorData then
            self.dataReady = true
            self.stamp = self.stamp + 1
        else
            print("message dropped")
        end
    end
end
```

```
--
-- Get current stamp (incremented after data is ready on each
iteration).
--
-- @return Number
--
SensorDataProcessor.getStamp = function(self)
    return self.stamp
end

--
-- Factory function, creates a mean SensorDataProcessor.
--
-- @param properties Table of input parameters
-- @param properties.parts String identifier for main data streams
(arm, elbow, hand)
-- @param properties.subparts String identifier for secondary data
streams (X, Y, Z)
-- @param properties.hexValues Number of hexadecimal values contained
in a single data stream
-- @param properties.hexSize Number of characters a hexValue is
comprised of
--
-- @return Table of output parameters
-- @return Table.evaluateCondition Function, see docs below
-- @return Table.process Function, see docs below
--
local createMeanProcessor = function(properties)
    -- identifier for invalid input values
    local invalidValue = tonumber(string.rep("F", properties.hexSize),
16)
    --local separators = {"\r\n", "\n", "\r"}
    --local storage = {}

    --
    -- Splits input into lines.
    --
    -- @param str String of raw data, contains newline characters.
    --
    -- @return Table
    --
    local extractMessages = function(str)
        local lines = {}

        for line in string.gmatch(str, "%S+") do
            table.insert(lines, line)
        end

        return lines
    end

    --
    -- Groups (by part/subpart) and sorts input data table.
    --
    -- @param lines Table of full message data (parts + subparts)
    --
    local preprocessMessages = function(lines)
        local temp = {}
        local i = 1
```

```
        local nparts, nsubparts = #properties.parts,
#properties.subparts

        if #lines < nparts * (nsubparts + 1) then return end

        -- always assume the following order:
        --  part {I, J, K}
        --  subpart X
        --  subpart Y
        --  subpart Z
        while i <= #lines do
            local ttemp = {} -- part + 3 subparts
            table.insert(ttemp, lines[i])
            i = i + 1

            for j = 1, nsubparts do
                table.insert(ttemp, lines[i])
                i = i + 1
            end

            table.insert(temp, ttemp)
        end

        -- sort alphabetically by first element, i.e. the part ("I",
"J", "K")
        table.sort(temp, function(a, b)
            return a[1] < b[1]
        end)

        -- clear original table
        for i, v in ipairs(lines) do lines[i] = nil end

        -- fill with correct sequence of parts and subparts, one level
of depth
        -- remove duplicate parts:
https://stackoverflow.com/a/12397571
        i = 1
        while i <= #temp do
            if i ~= 1 and temp[i][1] == temp[i - 1][1] then
                table.remove(temp, i)
            else
                for j, group in ipairs(temp[i]) do
                    table.insert(lines, group)
                end
                i = i + 1
            end
        end
    end

    --
    -- Tokenizes input string into elements of constant width.
    --
    -- @param str input String
    -- @param n Number of characters each token is comprised of
    --
    -- @return Table of split tokens
    --
    local splitString = function(str, n)
        n = math.floor(n or 0)
        if n <= 0 then return {""} end
        local t = {}
```

```
        for i = 1, math.floor(str:len() / n) do
            local start = 1 + n * (i - 1)
            table.insert(t, str:sub(start, start + n - 1))
        end

        return t
    end

    --
    -- Calculates arithmetic mean of input data.
    --
    -- @param varargs of input Numbers
    --
    -- @return Number as arithmetic mean of input data
    --
    local calculateMean = function(...)
        if select('#', ...) == 0 then return 0 end
        local sum = 0

        for i, value in ipairs({...}) do
            sum = sum + value
        end

        return math.floor(sum / select('#', ...))
    end

    --
    -- Extracts sensor value from sanitized streams and applies
selected algorithm.
    --
    -- @param Table of preprocessed message Strings.
    --
    -- @return Table on success, nil on failure
    --
    local doWork = function(lines)
        local nline = 0
        local storage = {}

        for i, part in ipairs(properties.parts) do
            nline = nline + 1
            local line = lines[nline]
            if not line or line ~= part then return nil end
            local partArray = {}

            for j, subpart in ipairs(properties.subparts) do
                nline = nline + 1
                line = lines[nline]
                if not line or line:sub(1, 1) ~= subpart then return
nil end
                local substring = line:sub(2)
                if substring:len() ~= properties.hexValues *
properties.hexSize then return nil end
                local thex = splitString(substring,
properties.hexSize)
                if #thex ~= properties.hexValues then return nil end

                for k, hexString in ipairs(thex) do
                    local dec = tonumber(hexString, 16)
                    if not dec then return nil end
                    if dec == invalidValue then dec = 0 end
```

```
                    local tuple = partArray[k] or {}
                    table.insert(tuple, dec)

                    if not partArray[k] then
                        table.insert(partArray, tuple)
                    end
                end
            end

            for k, tuple in ipairs(partArray) do
                partArray[k] = calculateMean(unpack(tuple))
            end

            table.insert(storage, partArray)
        end

        return storage
    end

    --
    -- Tests whether provided string represents a complete stream of
data (all parts and subparts).
    --
    -- @param str input String
    --
    -- @return Boolean
    --
    local evaluateCondition = function(str)
        local temp = {}
        local occurrences = 0
        local singleOccurrenceParts = {}

        for i, id in ipairs(properties.parts) do
            local j = 1

            -- find all occurrences of part 'id'
            while j <= str:len() do
                local n
                n, j = str:find(id, j, true)

                -- no more occurrences, exit loop
                if not n then break end

                -- store index in associative array for given part
                local t = temp[id] or {}
                table.insert(t, n)
                temp[id] = t

                j = j + 1
            end

            -- a part identifier is missing
            if not temp[id] then return false end

            -- if this part has a single occurrence, store its id
            if #temp[id] == 1 then
                table.insert(singleOccurrenceParts, id)
            end

            occurrences = occurrences + #temp[id]
        end
```

```
        -- must have at least nшu of parts + 1
        if not (occurrences > #properties.parts) then return false end

        local lastOccurrencePos = 0

        -- find position of last occurrence
        for id, ocs in pairs(temp) do
            for j, n in ipairs(ocs) do
                if n > lastOccurrencePos then lastOccurrencePos = n
end
            end
        end

        -- make sure that a part with a single occurrence is not last
        for i, id in ipairs(singleOccurrenceParts) do
            if temp[id][1] == lastOccurrencePos then return false end
        end

        return true
    end

    --
    -- Extract the biggest chunk of text containing full part frames.
    --
    -- @param str input string
    --
    -- @return desired String chunk
    -- @return rightmost remainder String chunk
    --
    local extractSubstring = function(str)
        local first, last = str:len(), str:len()

        for i, part in ipairs(properties.parts) do
            local f = str:find(part, 1, true)
            if f < first then first = f end
            local l = str:reverse():find(part, 1, true)
            if l < last then last = l end
        end

        last = str:len() - last
        return str:sub(first, last), str:sub(last + 1)
    end

    --
    -- Main execution block, processes raw data.
    --
    -- @param str String of raw data accumulated by the processor in
previous iterations
    --
    -- @return Table of parsed messages
    -- @return String of unprocessed data left by this iteration
    --
    local process = function(str)
        local substring, remainder = extractSubstring(str)
        local lines = extractMessages(substring)
        preprocessMessages(lines)
        return doWork(lines), remainder
    end

    return {
```

```
        evaluateCondition = evaluateCondition,
        process = process
    }
end

local sensorDataProcessor

--
-- Method called when the port monitor is created.
--
-- @param options
--
-- @return Boolean
--
PortMonitor.create = function(options)
    print("INITIALIZING AMOR SENSORS")
    -- TODO: read options from .ini file
    local meanProcessor = createMeanProcessor{
        parts = {"I", "J", "K"},
        subparts = {"X", "Y", "Z"},
        hexValues = 16,
        hexSize = 3
    }
    sensorDataProcessor = SensorDataProcessor:new(meanProcessor)
    return true;
end

--
-- Method called when port monitor is destroyed.
--
PortMonitor.destroy = function()
    print("DESTROYING PORT MONITOR")
    sensorDataProcessor = nil
end

--
-- Method called when the port receives new data.
-- If false is returned, the data will be ignored
-- and update() will never be called.
--
-- @param thing The Things abstract data type
--
-- @return Boolean
--
PortMonitor.accept = function(thing)
    bt = thing:asBottle()

    if bt == nil then
        print("bot_modifier.lua: got wrong data type (expected type
Bottle)")
        return false
    end

    sensorDataProcessor:accept(bt:get(0):asString())
    return sensorDataProcessor.dataReady
end

--
-- Method called to forward processed data.
--
-- @param thing The Things abstract data type
```

```
--
-- @return Things
--
PortMonitor.update = function(thing)
    print(string.format("in update [%d]",
sensorDataProcessor:getStamp()))
    sensorDataProcessor.dataReady = false
    local vec = yarp.Vector()

    for i, part in ipairs(sensorDataProcessor.currentSensorData) do
        for j, decValue in ipairs(part) do
            vec:push_back(decValue)
        end
    end

    thing:setPortWriter(vec)
    return thing
end
```

Anexo 6. Sensor_reader.py

```
import yarp
import Tkinter as tk

yarp.Network.init()

root = tk.Tk()

armSensors = tk.LabelFrame(root, text="Arm sensors")
armSensors.pack(fill="both", expand="yes", side="left")

forearmSensors = tk.LabelFrame(root, text="Forearm sensors")
forearmSensors.pack(fill="both", expand="yes", side="left")

handSensors = tk.LabelFrame(root, text="Hand sensors")
handSensors.pack(fill="both", expand="yes", side="left")

sa1 = tk.Label(armSensors, text="s1: ")
sa1.pack()

sa2 = tk.Label(armSensors, text="s2: ")
sa2.pack()

sa3 = tk.Label(armSensors, text="s3: ")
sa3.pack()

sa4 = tk.Label(armSensors, text="s4: ")
sa4.pack()

sa5 = tk.Label(armSensors, text="s5: ")
sa5.pack()

sa6 = tk.Label(armSensors, text="s6: ")
sa6.pack()

sa7 = tk.Label(armSensors, text="s7: ")
sa7.pack()

sa8 = tk.Label(armSensors, text="s8: ")
sa8.pack()

sa9 = tk.Label(armSensors, text="s9: ")
sa9.pack()

sa10 = tk.Label(armSensors, text="s10: ")
sa10.pack()

sa11 = tk.Label(armSensors, text="s11: ")
sa11.pack()

sa12 = tk.Label(armSensors, text="s12: ")
sa12.pack()

sa13 = tk.Label(armSensors, text="s13: ")
sa13.pack()

sa14 = tk.Label(armSensors, text="s14: ")
sa14.pack()
```

```
sa15 = tk.Label(armSensors, text="s15: ")
sa15.pack()

sa16 = tk.Label(armSensors, text="s16: ")
sa16.pack()

sf1 = tk.Label(forearmSensors, text="s1: ")
sf1.pack()

sf2 = tk.Label(forearmSensors, text="s2: ")
sf2.pack()

sf3 = tk.Label(forearmSensors, text="s3: ")
sf3.pack()

sf4 = tk.Label(forearmSensors, text="s4: ")
sf4.pack()

sf5 = tk.Label(forearmSensors, text="s5: ")
sf5.pack()

sf6 = tk.Label(forearmSensors, text="s6: ")
sf6.pack()

sf7 = tk.Label(forearmSensors, text="s7: ")
sf7.pack()

sf8 = tk.Label(forearmSensors, text="s8: ")
sf8.pack()

sf9 = tk.Label(forearmSensors, text="s9: ")
sf9.pack()

sf10 = tk.Label(forearmSensors, text="s10: ")
sf10.pack()

sf11 = tk.Label(forearmSensors, text="s11: ")
sf11.pack()

sf12 = tk.Label(forearmSensors, text="s12: ")
sf12.pack()

sf13 = tk.Label(forearmSensors, text="s13: ")
sf13.pack()

sf14 = tk.Label(forearmSensors, text="s14: ")
sf14.pack()

sf15 = tk.Label(forearmSensors, text="s15: ")
sf15.pack()

sf16 = tk.Label(forearmSensors, text="s16: ")
sf16.pack()

sh1 = tk.Label(handSensors, text="s1: ")
sh1.pack()

sh2 = tk.Label(handSensors, text="s2: ")
sh2.pack()
```

```
sh3 = tk.Label(handSensors, text="s3: ")
sh3.pack()

sh4 = tk.Label(handSensors, text="s4: ")
sh4.pack()

sh5 = tk.Label(handSensors, text="s5: ")
sh5.pack()

sh6 = tk.Label(handSensors, text="s6: ")
sh6.pack()

sh7 = tk.Label(handSensors, text="s7: ")
sh7.pack()

sh8 = tk.Label(handSensors, text="s8: ")
sh8.pack()

sh9 = tk.Label(handSensors, text="s9: ")
sh9.pack()

sh10 = tk.Label(handSensors, text="s10: ")
sh10.pack()

sh11 = tk.Label(handSensors, text="s11: ")
sh11.pack()

sh12 = tk.Label(handSensors, text="s12: ")
sh12.pack()

sh13 = tk.Label(handSensors, text="s13: ")
sh13.pack()

sh14 = tk.Label(handSensors, text="s14: ")
sh14.pack()

sh15 = tk.Label(handSensors, text="s15: ")
sh15.pack()

sh16 = tk.Label(handSensors, text="s16: ")
sh16.pack()

class DataProcessor(yarp.PortReader):
    def read(self, connection):
        print("in DataProcessor.read")
        if not connection.isValid():
            print("Connection shutting down")
            return False
        b = yarp.Bottle()
        print("Trying to read from connection")
        ok = b.read(connection)
        if not ok:
            print("Failed to read input")
            return False
        print("Received [%s]"%b.toString())
        sh1['text'] = 's1: %d' % b.get(0).asDouble()
        sh2['text'] = 's2: %d' % b.get(1).asDouble()
        sh3['text'] = 's3: %d' % b.get(2).asDouble()
        sh4['text'] = 's4: %d' % b.get(3).asDouble()
        sh5['text'] = 's5: %d' % b.get(4).asDouble()
        sh6['text'] = 's6: %d' % b.get(5).asDouble()
```

```
        sh7['text'] = 's7: %d' % b.get(6).asDouble()
        sh8['text'] = 's8: %d' % b.get(7).asDouble()
        sh9['text'] = 's9: %d' % b.get(8).asDouble()
        sh10['text'] = 's10: %d' % b.get(9).asDouble()
        sh11['text'] = 's11: %d' % b.get(10).asDouble()
        sh12['text'] = 's12: %d' % b.get(11).asDouble()
        sh13['text'] = 's13: %d' % b.get(12).asDouble()
        sh14['text'] = 's14: %d' % b.get(13).asDouble()
        sh15['text'] = 's15: %d' % b.get(14).asDouble()
        sh16['text'] = 's16: %d' % b.get(15).asDouble()
        sf1['text'] = 's1: %d' % b.get(16).asDouble()
        sf2['text'] = 's2: %d' % b.get(17).asDouble()
        sf3['text'] = 's3: %d' % b.get(18).asDouble()
        sf4['text'] = 's4: %d' % b.get(19).asDouble()
        sf5['text'] = 's5: %d' % b.get(20).asDouble()
        sf6['text'] = 's6: %d' % b.get(21).asDouble()
        sf7['text'] = 's7: %d' % b.get(22).asDouble()
        sf8['text'] = 's8: %d' % b.get(23).asDouble()
        sf9['text'] = 's9: %d' % b.get(24).asDouble()
        sf10['text'] = 's10: %d' % b.get(25).asDouble()
        sf11['text'] = 's11: %d' % b.get(26).asDouble()
        sf12['text'] = 's12: %d' % b.get(27).asDouble()
        sf13['text'] = 's13: %d' % b.get(28).asDouble()
        sf14['text'] = 's14: %d' % b.get(29).asDouble()
        sf15['text'] = 's15: %d' % b.get(30).asDouble()
        sf16['text'] = 's16: %d' % b.get(31).asDouble()
        sa1['text'] = 's1: %d' % b.get(32).asDouble()
        sa2['text'] = 's2: %d' % b.get(33).asDouble()
        sa3['text'] = 's3: %d' % b.get(34).asDouble()
        sa4['text'] = 's4: %d' % b.get(35).asDouble()
        sa5['text'] = 's5: %d' % b.get(36).asDouble()
        sa6['text'] = 's6: %d' % b.get(37).asDouble()
        sa7['text'] = 's7: %d' % b.get(38).asDouble()
        sa8['text'] = 's8: %d' % b.get(39).asDouble()
        sa9['text'] = 's9: %d' % b.get(40).asDouble()
        sa10['text'] = 's10: %d' % b.get(41).asDouble()
        sa11['text'] = 's11: %d' % b.get(42).asDouble()
        sa12['text'] = 's12: %d' % b.get(43).asDouble()
        sa13['text'] = 's13: %d' % b.get(44).asDouble()
        sa14['text'] = 's14: %d' % b.get(45).asDouble()
        sa15['text'] = 's15: %d' % b.get(46).asDouble()
        sa16['text'] = 's16: %d' % b.get(47).asDouble()
        return True

p = yarp.Port()
r = DataProcessor()
p.setReader(r)
p.open("/sensor_reader");

root.mainloop()

p.close()

yarp.Network.fini();
```

Anexo 7. Main.cpp

```
//A serial class that appears as an iostream

#include <iostream>
#include "serialstream.h"
#include <yarp\os\all.h>


using namespace std;
using namespace boost::posix_time;
using namespace yarp::os;


void mostrarSensores1(char Sensores [3][3][16][3]);
void mostrarSensores2(char Sensores [3][3][16][3]);
int CharHex3ToIntDec (char a, char b, char c);
void mostrarSensores3(int Sensores [3][3][16]);
void mostrarSensores4(int Sensores [3][3][16]);
void mostrarSensoresMedia(int Sensores [3][3][16]);
int media3(int a, int b, int c);
//void inicializarMatrizSensores(char Sensores [3][3][16][3])



int main(int argc, char* argv[])
{
	Network Yarp;
	BufferedPort<Bottle> out;
	out.open("/outSensor");

	SerialOptions options;
	options.setDevice("COM3");
	options.setBaudrate(115200);
	options.setTimeout(seconds(30));
	//options.setFlowControl(SerialOptions::sofware);
	//options.setParity(SerialOptions::even);
	//options.setCsize(5);
	SerialStream serial(options);
	serial.exceptions(ios::badbit | ios::failbit); //Important!
	//serial<<"Hello world"<<endl;
	serial.clear();

	try {

		string s="0";
		int Sensores [3][3][16]; //[Zona del
brazo][Led1,2,3][N°deSensor]
		for (int l=0;l<3;l++){
			for (int k=0;k<3;k++){
				for (int j=0;j<16;j++){
					Sensores [l][k][j]=0;
				}
			}
		}
		bool Ix=0,Iy=0,Iz=0,Jx=0,Jy=0,Jz=0,Kx=0,Ky=0,Kz=0;
		bool Iok=0, Jok=0, Kok=0;
		//serial>>s;
		while (s!="OK"){
```

```
                  serial>>s;
                  //cout<<s<<endl;
            }
            while (1){

                  serial>>s;
                  //cout<<s<<endl;
                  //serial.clear();

                  //cout<<"Buscando I"<<endl;
                  if (s[0]=='I') {
                        //cout<<"detectada I"<<endl;
                        serial>>s;
                        //cout<<s<<endl;
                        if (s.size()==49){
                              if (s[0]=='X'){
                                    //cout<<"detectada X"<<endl;
                                    for (int j=0;j<16;j++){

                                          //cout<<"Leyendo el valor
"<<j<<"de X"<<endl;

      Sensores[0][0][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                          Ix=1;
                                    }
                              }
                        }
                        serial>>s;
                        //cout<<s<<endl;
                        if (s.size()==49){
                              if (s[0]=='Y'){
                                    //cout<<"detectada Y"<<endl;

                                    for (int j=0;j<16;j++){
                                          //cout<<"Leyendo el valor
"<<j<<"de Y"<<endl;

      Sensores[0][1][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                          Iy=1;
                                    }
                              }
                        }
                        serial>>s;
                        //cout<<s<<endl;
                        if (s.size()==49){
                              if (s[0]=='Z'){
                                    //cout<<"detectada Z"<<endl;
                                    for (int j=0;j<16;j++){
                                          //cout<<"Leyendo el valor
"<<j<<"de Z"<<endl;

      Sensores[0][2][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                          Iz=1;
                                    }
                              }
                        }
                        if(Ix==1&&Iy==1&&Iz==1){
                              Iok=1;
```

```
                                    //cout<<"Iok"<<endl;
                                    Ix=0;
                                    Iy=0;
                                    Iz=0;
                              }
                        }

                        //cout<<"Buscando J"<<endl;
                        if (s[0]=='J') {
                              //cout<<"detectada J"<<endl;
                              serial>>s;
                              //cout<<s<<endl;
                              if (s.size()==49){
                                    if (s[0]=='X'){
                                          //cout<<"detectada X"<<endl;

                                          for (int j=0;j<16;j++){

                                                //cout<<"Leyendo el valor
"<<j<<"de X"<<endl;

      Sensores[1][0][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                                Jx=1;
                                          }
                                    }
                              }
                              serial>>s;
                              //cout<<s<<endl;
                              if (s.size()==49){
                                    if (s[0]=='Y'){
                                          //cout<<"detectada Y"<<endl;

                                          for (int j=0;j<16;j++){
                                                //cout<<"Leyendo el valor
"<<j<<"de Y"<<endl;

      Sensores[1][1][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                                Jy=1;
                                          }
                                    }
                              }
                              serial>>s;
                              //cout<<s<<endl;
                              if (s.size()==49){
                                    if (s[0]=='Z'){
                                          //cout<<"detectada Z"<<endl;
                                          for (int j=0;j<16;j++){
                                                //cout<<"Leyendo el valor
"<<j<<"de Z"<<endl;

      Sensores[1][2][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                                Jz=1;
                                          }
                                    }
                              }
                              if(Jx==1&&Jy==1&&Jz==1){
                                    Jok=1;
                                    //cout<<"Jok"<<endl;
```

```
                              Jx=0;
                              Jy=0;
                              Jz=0;
                        }
                  }

                  //cout<<"Buscando K"<<endl;
                  if (s[0]=='K') {
                        //cout<<"detectada K"<<endl;
                        serial>>s;
                        //cout<<s<<endl;
                        if (s.size()==49){
                              if (s[0]=='X'){
                                    //cout<<"detectada X"<<endl;
                                    for (int j=0;j<16;j++){

                                          //cout<<"Leyendo el valor
"<<j<<"de X"<<endl;

      Sensores[2][0][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                          Kx=1;
                                    }
                              }
                        }
                        serial>>s;
                        //cout<<s<<endl;
                        if (s.size()==49){
                              if (s[0]=='Y'){
                                    //cout<<"detectada Y"<<endl;

                                    for (int j=0;j<16;j++){
                                          //cout<<"Leyendo el valor
"<<j<<"de Y"<<endl;

      Sensores[2][1][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                          Ky=1;
                                    }
                              }
                        }
                        serial>>s;
                        //cout<<s<<endl;
                        if (s.size()==49){
                              if (s[0]=='Z'){
                                    //cout<<"detectada Z"<<endl;
                                    for (int j=0;j<16;j++){
                                          //cout<<"Leyendo el valor
"<<j<<"de Z"<<endl;

      Sensores[2][2][j]=CharHex3ToIntDec(s[(j*3)+1],s[(j*3)+2],s[(j*3)
+3]);
                                          Kz=1;
                                    }
                              }
                        }
                        if(Kx==1&&Ky==1&&Kz==1){
                              Kok=1;
                              //cout<<"Kok"<<endl;
                              Kx=0;
                              Ky=0;
```

```
                                    Kz=0;
                              }
                        }

                        serial.clear();

                        if ((Iok==1)&&(Jok==1)&&(Kok==1)){
                              mostrarSensoresMedia(Sensores);
                              Iok=0;
                              Jok=0;
                              Kok=0;

                                    //Yarp bottle's preparation
                              Bottle& bottle_sensor= out.prepare();
                              bottle_sensor.clear();  //Esto se me olvidaba
ponerlo y escribia cada vez una botella más larga.
                              bottle_sensor.addString("i01");

      bottle_sensor.addInt(media3(Sensores[0][0][0],Sensores[0][1][0],
Sensores[0][2][0]));
                              bottle_sensor.addString("i02");

      bottle_sensor.addInt(media3(Sensores[0][0][2],Sensores[0][1][2],
Sensores[0][2][2]));
                              bottle_sensor.addString("i03");

      bottle_sensor.addInt(media3(Sensores[0][0][3],Sensores[0][1][3],
Sensores[0][2][3]));
                              bottle_sensor.addString("i04");

      bottle_sensor.addInt(media3(Sensores[0][0][4],Sensores[0][1][4],
Sensores[0][2][4]));
                              bottle_sensor.addString("i05");

      bottle_sensor.addInt(media3(Sensores[0][0][5],Sensores[0][1][5],
Sensores[0][2][5]));
                              bottle_sensor.addString("i06");

      bottle_sensor.addInt(media3(Sensores[0][0][6],Sensores[0][1][6],
Sensores[0][2][6]));
                              bottle_sensor.addString("i07");

      bottle_sensor.addInt(media3(Sensores[0][0][7],Sensores[0][1][7],
Sensores[0][2][7]));
                              bottle_sensor.addString("i08");

      bottle_sensor.addInt(media3(Sensores[0][0][8],Sensores[0][1][8],
Sensores[0][2][8]));
                              bottle_sensor.addString("i09");

      bottle_sensor.addInt(media3(Sensores[0][0][9],Sensores[0][1][9],
Sensores[0][2][9]));
                              bottle_sensor.addString("i10");

      bottle_sensor.addInt(media3(Sensores[0][0][10],Sensores[0][1][10
],Sensores[0][2][10]));
                              bottle_sensor.addString("i11");

      bottle_sensor.addInt(media3(Sensores[0][0][11],Sensores[0][1][11
],Sensores[0][2][11]));
                              bottle_sensor.addString("i12");
```

```
      bottle_sensor.addInt(media3(Sensores[0][0][12],Sensores[0][1][12
],Sensores[0][2][12]));
                          bottle_sensor.addString("i13");

      bottle_sensor.addInt(media3(Sensores[0][0][13],Sensores[0][1][13
],Sensores[0][2][13]));
                          bottle_sensor.addString("j01");

      bottle_sensor.addInt(media3(Sensores[1][0][0],Sensores[1][1][0],
Sensores[1][2][0]));
                          bottle_sensor.addString("j02");

      bottle_sensor.addInt(media3(Sensores[1][0][1],Sensores[1][1][1],
Sensores[1][2][1]));
                          bottle_sensor.addString("j03");

      bottle_sensor.addInt(media3(Sensores[1][0][2],Sensores[1][1][2],
Sensores[1][2][2]));
                          bottle_sensor.addString("j04");

      bottle_sensor.addInt(media3(Sensores[1][0][3],Sensores[1][1][3],
Sensores[1][2][3]));
                          bottle_sensor.addString("j05");

      bottle_sensor.addInt(media3(Sensores[1][0][4],Sensores[1][1][4],
Sensores[1][2][4]));
                          bottle_sensor.addString("j06");

      bottle_sensor.addInt(media3(Sensores[1][0][5],Sensores[1][1][5],
Sensores[1][2][5]));
                          bottle_sensor.addString("j07");

      bottle_sensor.addInt(media3(Sensores[1][0][6],Sensores[1][1][6],
Sensores[1][2][6]));
                          bottle_sensor.addString("j08");

      bottle_sensor.addInt(media3(Sensores[1][0][9],Sensores[1][1][9],
Sensores[1][2][9]));
                          bottle_sensor.addString("j09");

      bottle_sensor.addInt(media3(Sensores[1][0][10],Sensores[1][1][10
],Sensores[1][2][10]));
                          bottle_sensor.addString("j10");

      bottle_sensor.addInt(media3(Sensores[1][0][11],Sensores[1][1][11
],Sensores[1][2][11]));
                          bottle_sensor.addString("j11");

      bottle_sensor.addInt(media3(Sensores[1][0][12],Sensores[1][1][12
],Sensores[1][2][12]));
                          bottle_sensor.addString("j12");

      bottle_sensor.addInt(media3(Sensores[1][0][13],Sensores[1][1][13
],Sensores[1][2][13]));
                          bottle_sensor.addString("j13");

      bottle_sensor.addInt(media3(Sensores[1][0][14],Sensores[1][1][14
],Sensores[1][2][14]));
                          bottle_sensor.addString("j14");
```

```
      bottle_sensor.addInt(media3(Sensores[1][0][15],Sensores[1][1][15
],Sensores[1][2][15]));
                           bottle_sensor.addString("k01");

      bottle_sensor.addInt(media3(Sensores[2][0][0],Sensores[2][1][0],
Sensores[2][2][0]));
                           bottle_sensor.addString("k02");

      bottle_sensor.addInt(media3(Sensores[2][0][1],Sensores[2][1][1],
Sensores[2][2][1]));
                           bottle_sensor.addString("k03");

      bottle_sensor.addInt(media3(Sensores[2][0][2],Sensores[2][1][2],
Sensores[2][2][2]));
                           bottle_sensor.addString("k04");

      bottle_sensor.addInt(media3(Sensores[2][0][3],Sensores[2][1][3],
Sensores[2][2][3]));
                           bottle_sensor.addString("k05");

      bottle_sensor.addInt(media3(Sensores[2][0][4],Sensores[2][1][4],
Sensores[2][2][4]));
                           bottle_sensor.addString("k06");

      bottle_sensor.addInt(media3(Sensores[2][0][5],Sensores[2][1][5],
Sensores[2][2][5]));
                           bottle_sensor.addString("k07");

      bottle_sensor.addInt(media3(Sensores[2][0][6],Sensores[2][1][6],
Sensores[2][2][6]));
                           bottle_sensor.addString("k08");

      bottle_sensor.addInt(media3(Sensores[2][0][7],Sensores[2][1][7],
Sensores[2][2][7]));
                           bottle_sensor.addString("k09");

      bottle_sensor.addInt(media3(Sensores[2][0][8],Sensores[2][1][8],
Sensores[2][2][8]));
                           bottle_sensor.addString("k10");

      bottle_sensor.addInt(media3(Sensores[2][0][9],Sensores[2][1][9],
Sensores[2][2][9]));
                           bottle_sensor.addString("k11");

      bottle_sensor.addInt(media3(Sensores[2][0][10],Sensores[2][1][10
],Sensores[2][2][10]));
                           bottle_sensor.addString("k12");

      bottle_sensor.addInt(media3(Sensores[2][0][11],Sensores[2][1][11
],Sensores[2][2][11]));
                           bottle_sensor.addString("k13");

      bottle_sensor.addInt(media3(Sensores[2][0][12],Sensores[2][1][12
],Sensores[2][2][12]));
                           bottle_sensor.addString("k14");

      bottle_sensor.addInt(media3(Sensores[2][0][13],Sensores[2][1][13
],Sensores[2][2][13]));
                           bottle_sensor.addString("k15");
```

```
      bottle_sensor.addInt(media3(Sensores[2][0][14],Sensores[2][1][14
],Sensores[2][2][14]));
                        bottle_sensor.addString("k16");

      bottle_sensor.addInt(media3(Sensores[2][0][15],Sensores[2][1][15
],Sensores[2][2][15]));
                        out.write();

                  }


            }
      } catch(TimeoutException&) {
            serial.clear(); //Don't forget to clear error flags after a
timeout
            cerr<<"Timeout occurred"<<endl;
      }
      return 0;
}
/*void inicializarMatrizSensores(char Sensores [3][3][16][3]){
for (int l=0;l<3;l++){
for (int k=0;k<3;k++){
for (int j=0;j<16;j++){
for (int i=0;i<3;i++){
Sensores [l][k][j][i]='X';
}
}
}
}
}*/

void mostrarSensores1(char Sensores [3][3][16][3]) {
      for (int j=0;j<16;j++){   //Mostrar sensores 1
            cout<<"  "<<j+1<<endl;
            for(int k=0;k<3;k++){
                  for (int i=0;i<3;i++){
                        cout<<"  "<<Sensores[0][k][j][i];
                  }
                  cout<<endl;
            }
            cout<<endl;
      }
}
void mostrarSensores2(char Sensores [3][3][16][3]) {

      cout<<" ";
      for (int j=0;j<9;j++){
            cout<<j+1<<"   ";
      }
      for (int j=9;j<16;j++){
            cout<<j+1<<"  ";
      }
      cout<<endl;
      for (int k=0;k<3;k++){
            for (int j=0;j<16;j++){
                  for (int i=0;i<3;i++){
                        cout<<Sensores[0][k][j][i];
                  }
                  cout<<" ";
            }
```

```
            cout<<endl;
      }
      cout<<endl;
}
void mostrarSensores3(int Sensores [3][3][16]){

      cout<<" ";
      for (int j=0;j<9;j++){
            cout<<j+1<<"    ";
      }
      for (int j=9;j<16;j++){
            cout<<j+1<<"   ";
      }
      cout<<endl;
      for (int k=0;k<3;k++){
            for (int j=0;j<16;j++){
                  cout<<Sensores[0][k][j];
                  cout<<" ";
            }
            cout<<endl;
      }
      cout<<endl;

}
void mostrarSensores4(int Sensores [3][3][16]){

      cout<<"Mano:"<<endl;
      for (int j=0;j<16;j++){
            cout<<j+1<<"    "<<Sensores[0][0][j]<<"
"<<Sensores[0][1][j]<<"   "<<Sensores[0][2][j]<<"   "<<endl;
      }
      cout<<"Antebrazo:"<<endl;
      for (int j=0;j<16;j++){
            cout<<j+1<<"    "<<Sensores[1][0][j]<<"
"<<Sensores[1][1][j]<<"   "<<Sensores[1][2][j]<<"   "<<endl;
      }
      cout<<"Brazo:"<<endl;
      for (int j=0;j<16;j++){
            cout<<j+1<<"    "<<Sensores[2][0][j]<<"
"<<Sensores[2][1][j]<<"   "<<Sensores[2][2][j]<<"   "<<endl;
      }
}

void mostrarSensoresMedia(int Sensores[3][3][16]){

      cout<<"Mano:"<<endl;
      for (int j=0;j<16;j++){
            cout<<j+1<<"
"<<media3(Sensores[0][0][j],Sensores[0][1][j],Sensores[0][2][j])<<endl
;
      }
      cout<<"Antebrazo:"<<endl;
      for (int j=0;j<16;j++){
            cout<<j+1<<"
"<<media3(Sensores[1][0][j],Sensores[1][1][j],Sensores[1][2][j])<<endl
;
      }
      cout<<"Brazo:"<<endl;
      for (int j=0;j<16;j++){
```

```
            cout<<j+1<<"
"<<media3(Sensores[2][0][j],Sensores[2][1][j],Sensores[2][2][j])<<endl
;
      }
}
int CharHex3ToIntDec (char a, char b, char c){

      long int dec=0;

      if
((a=='0')||(a=='1')||(a=='2')||(a=='3')||(a=='4')||(a=='5')||(a=='6')|
|(a=='7')||(a=='8')||(a=='9')){

            dec+=((a-48)*256); //Tercer valor del HEX. El -48 es porque
el valor dec del char equivale al numero que representa en ascii menos
48 (char 0 = Dec 48)
      }
      else if ((a=='A')||(a=='B')||(a=='C')||(a=='D')||(a=='E')){

            dec+=((a-55)*256); //porque el valor dec del char equivale
a la letra que representa en ascii menos 55 (char A = Dec 65)
      }
      else if (a=='F'){ //Significa que el sensor esta desactivado.
            return 9999;
      }
      else { cout << "Error en conversion de Hex a Dec"<<endl;
      return 6666;
      }

      if
((b=='0')||(b=='1')||(b=='2')||(b=='3')||(b=='4')||(b=='5')||(b=='6')|
|(b=='7')||(b=='8')||(b=='9')){
            dec+=((b-48)*16); //Segundo valor del Hex
      }
      else if
((b=='A')||(b=='B')||(b=='C')||(b=='D')||(b=='E')||(b=='F')){
            dec+=((b-55)*16);
      }
      else { cout << "Error en conversion de Hex a Dec"<<endl;
      return 6666;
      }

      if
((c=='0')||(c=='1')||(c=='2')||(c=='3')||(c=='4')||(c=='5')||(c=='6')|
|(c=='7')||(c=='8')||(c=='9')){
            dec+=(c-48); //primer valor del Hex
      }
      else if
((c=='A')||(c=='B')||(c=='C')||(c=='D')||(c=='E')||(c=='F')){
            dec+=(c-55);
      }
      else { cout << "Error en conversion de Hex a Dec"<<endl;
      return 6666;
      }
      return dec;

}
int media3(int a, int b, int c){
      return (a+b+c)/3;
}
```

Anexo 8. DeviceDriver.cpp

```
// -*- mode:C++; tab-width:4; c-basic-offset:4; indent-tabs-mode:nil -
*-

#include "ProximitySensorsClient.hpp"

#include <string>
#include <sstream>

#include <yarp/os/Value.h>

#include <ColorDebug.hpp>

// ------------------- DeviceDriver Related ------------------------
----------

bool roboticslab::ProximitySensorsClient::open(yarp::os::Searchable&
config)
{
    CD_INFO("Starting ProximitySensorsClient plugin.\n");
    CD_DEBUG("config: %s.\n", config.toString().c_str());

    std::string local = config.check("local",
yarp::os::Value(DEFAULT_LOCAL), "local port").asString();
    std::string remote = config.check("remote",
yarp::os::Value(DEFAULT_REMOTE), "remote port").asString();

    sr.setReference(this);
    sr.open(local);
    sr.useCallback();

    std::string carrier;

    if (config.check("usePortMonitor") ||
(config.check("portMonitorType") && config.check("portMonitorContext")
            && config.check("portMonitorFile")))
    {
        std::string pmType = config.check("portMonitorType",
yarp::os::Value(DEFAULT_PORTMONITOR_TYPE),
                "port monitor type").asString();
        std::string pmContext = config.check("portMonitorContext",
yarp::os::Value(DEFAULT_PORTMONITOR_CONTEXT),
                "port monitor context").asString();
        std::string pmFile = config.check("portMonitorFile",
yarp::os::Value(DEFAULT_PORTMONITOR_FILE),
                "port monitor file").asString();

        std::ostringstream oss;
        oss << "tcp+recv.portmonitor+type." << pmType << "+context."
<< pmContext << "+file." << pmFile;

        carrier = oss.str();

        CD_INFO("Using carrier: %s\n", carrier.c_str());
    }

    if (!yarp::os::Network::connect(remote, local, carrier))
//Conexión plug in (conexión autom a yarp-puerto serie arduinos)
```

```
    {
        CD_ERROR("Unable to connect to remote port \"%s\".\n",
remote.c_str());
        close();
        return false;
    }

    return true;
}

// ----------------------------------------------------------------------
----------

bool roboticslab::ProximitySensorsClient::close()
{
    CD_INFO("Closing ProximitySensorsClient plugin.\n");
    sr.disableCallback();
    sr.close();
    return true;
}

// ----------------------------------------------------------------------
----------
```

Anexo 9. Prueba_trayectoria_programada.cpp

```
#include <stdio.h>
// #include <signal.h> //Para usar el ctrl + c

#include <amor.h>
#include <cmath> //Para el valor absoluto
#include <yarp/os/all.h>
//#include <yarp/dev/all.h>
//#include <yarp/sig/Vector.h>

// bool keepRunning = true;

// void intHandler(int dummy)
// {
//    keepRunning = false;
// }

AMOR_VECTOR7 g_cartPositions;
AMOR_HANDLE g_hRobot;
AMOR_VECTOR7 g_jointPosition;
AMOR_VECTOR7 g_cartTargetPosition;
AMOR_VECTOR7 g_cartVelocity;
double g_path[3];
double g_Pnormalized;
bool g_target= false; // si el flag se activa, es que el sensor 15 de la mano
ha detectado la botella
bool flag= false; //Si se activa el flag, es que hay un obtáculo cerca
const double g_threshold=300; //objetivo a los 6 cm de distancia
const double g_hand_threshold=50;
const double g_tolerance=15; //1.5 cm para llegar al objetivo (esfera)


#define RAD2DEG(a)      (180.0*a/PI)
#define DEG2RAD(a)      (PI*a/180.0)
static const real PI  =
3.14159265358979323846264338327950288419716939937510582097494 4;
static const real c = 10;

const real INITIAL_CART_POSITION[AMOR_NUM_JOINTS] = {
      7.69, -419.72, 179.81, DEG2RAD(175.616), DEG2RAD(-90.216), DEG2RAD(-
2.712) };

const real TARGET_CART_POSITION[AMOR_NUM_JOINTS] = {
    7.69, -637.74, 179.81, DEG2RAD(175.616), DEG2RAD(-90.216), DEG2RAD(-
2.712) };


// FUNCTION DECLARATIONS

bool tolerance(AMOR_VECTOR7 g_cartPositions);

// FUNCTION DEFINITIONS

bool tolerance(AMOR_VECTOR7 g_cartPositions) //Si el valor de la tolerancia
es menor o igual que 2 cm para cada coordenada, consideramos que está en el
objetivo (true)
```

```
{
double tolerance[3];
for(int i=0;i<3;i++)
{tolerance[i]=std::abs(TARGET_CART_POSITION[i]-g_cartPositions[i]);}
return tolerance[0]<=g_tolerance && tolerance[1]<=g_tolerance &&
tolerance[2]<=g_tolerance;
}

class SensorReader : public yarp::os::BufferedPort<yarp::os::Bottle>
{
public:
    virtual void onRead(yarp::os::Bottle& b)
    {
       if(b.size()==0)
       return;
       if(b.get(14).asDouble() > g_hand_threshold && b.get(14).asDouble() <
1000)
         {g_target=true;
          printf("Botella detectada\n");}
       double max=0;
       for(int i=0;i<b.size();i++)
       {if(b.get(i).asDouble()>max)
       max=b.get(i).asDouble();}
         printf("Current maximum sensor value: %f\n", max);
       if(max>g_threshold)
       flag=true;
       else
       flag=false;
       //flag = max > g_threshold;


       //printf("%s\n", b.toString().c_str());
         //printf("%f\n", b.get(5).asDouble());
    }

};

int main(int argc, char *argv[])
{

yarp::os::Network::init();

SensorReader sr;
sr.open("/sensor_reader");
sr.useCallback();


#ifdef __linux__
            g_hRobot = amor_connect((char *)"libeddriver.so", 0);
#endif // LINUX

 if(g_hRobot == AMOR_INVALID_HANDLE)
       return 1;                          // if AMOR connected


//Comenzar en la posición inicial del robot

            printf("Going to initial position...\n");
            for(unsigned i = 0; i < 6; i++)
```

```
                g_cartPositions[i] = INITIAL_CART_POSITION[i];
            if(amor_set_cartesian_positions(g_hRobot, g_cartPositions) !=
AMOR_SUCCESS)
          printf("%s\n", amor_error());
              // HandleError();

      yarp::os::Time::delay(2); //retardo 2 segundos para que le de tiempo a
ejecutar la funcion


//Comprobar posición del robot

            printf("Posición actual:\n");
          amor_get_cartesian_position(g_hRobot, g_cartPositions);
            for(unsigned i = 0; i < 3; ++i)
                printf("%.2f ", g_cartPositions[i]);
            for(unsigned i = 3; i < 6; ++i)
                printf("%.3f ", RAD2DEG(g_cartPositions[i]));
            printf("\n");

//Hallar trayectoria del robot

      for(int i=0;i<3;i++)
      g_path[i]=TARGET_CART_POSITION[i] - g_cartPositions[i];

//Normalizar vector trayectoria

      g_Pnormalized=sqrt(g_path[0]*g_path[0]+g_path[1]*g_path[1]+g_path[2]*g
_path[2]);

//Obtener vector unitario

      if(g_Pnormalized>0)
      {for(int i=0;i<3;i++)
      {  g_path[i]/=g_Pnormalized;  }};

//Establecer velocidad del robot a partir de la trayectoria calculada
(VELOCIDAD NORMAL DE MOVIMIENTO, SIN OBSTÁCULOS)

      for(int i=0; i<3; i++)
            g_cartVelocity[i]=c*g_path[i];
      for(int i=0; i<6; i++)
            printf("%.2f ", g_cartVelocity[i]);
            printf("\n");
            yarp::os::Time::delay(5);
          if(amor_set_cartesian_velocities(g_hRobot, g_cartVelocity) !=
AMOR_SUCCESS)
            printf("%s\n", amor_error());


while(true)

{

      amor_get_cartesian_position(g_hRobot, g_cartPositions);
      for(int i=0; i<6; i++)
            printf("%.2f ", g_cartPositions[i]); //Para visualizar la
posición del robot en cada momento
            printf("\n");
```

```
      if(g_target || tolerance(g_cartPositions))
      break;
      if(flag)
      amor_controlled_stop(g_hRobot);
      else
      if(amor_set_cartesian_velocities(g_hRobot, g_cartVelocity) !=
AMOR_SUCCESS)
      printf("%s\n", amor_error());



/*Mover el robot hacia el objetivo (me muevo 10 cm en el eje z) ---PRUEBA DÍA
1

          g_cartPositions[2]+=100.92;
          if(amor_set_cartesian_positions(g_hRobot, g_cartPositions) !=
AMOR_SUCCESS)
             printf("%s\n", amor_error()); */

      //yarp::os::Time::delay(5); //retardo 5 segundos para que le de tiempo
a ejecutar la funcion


/*    enum
amor_movement_status{AMOR_MOVEMENT_STATUS_MOVING,AMOR_MOVEMENT_STATUS_PENDING
,AMOR_MOVEMENT_STATUS_FINISHED};
      amor_get_movement_status(AMOR_HANDLE handle, amor_movement_status*
status);
      /// AMOR is moving, movement is pending or reached the requested
position
*/
          yarp::os::Time::delay(0.05); //Dentro del bucle while para que no
se sature (delay en segundos)

//PARADA DE EMERGENCIA


} //cierro while true

sr.disableCallback(); //para que, una vez llegado al objetivo, deje de entrar
en la función.

g_cartPositions[0] = g_cartPositions[1] = g_cartPositions[2] = 0;
amor_set_cartesian_velocities(g_hRobot, g_cartVelocity);
amor_controlled_stop(g_hRobot);


//cierre de la garra
double now=yarp::os::Time::now(); //tiempo actual antes de entrar en el bucle
(necesito ejecutarla en bucle porque avanza de 1mm en 1mm aprox.)
while(yarp::os::Time::now()-now<4)
{amor_close_hand(g_hRobot);
}
amor_controlled_stop(g_hRobot);

            amor_release(g_hRobot);

   /*   Posición articular:
```

```
      if(amor_get_actual_positions(g_hRobot, &g_jointPosition) !=
AMOR_SUCCESS)
      return 1;
        for(unsigned i = 0; i < AMOR_NUM_JOINTS; ++i)
        printf("%.3f ", RAD2DEG(g_jointPosition[i]));
        printf("\n"); */



    sr.close();

    return 0;
}

//PARADAS DE EMERGENCIA

       // amor_emergency_stop(g_hRobot);
```

Anexo 10. ProximitySensorsClient.cpp

```
// -*- mode:C++; tab-width:4; c-basic-offset:4; indent-tabs-mode:nil -
*-

#include "ProximitySensorsClient.hpp"

#include <ColorDebug.hpp>

const int roboticslab::ProximitySensorsClient::THRESHOLD_GRIPPER = 50;
const int roboticslab::ProximitySensorsClient::THRESHOLD_ALERT = 800;
const int roboticslab::ProximitySensorsClient::THRESHOLD_LOW_ALERT =
100;

void
roboticslab::ProximitySensorsClient::SensorReader::onRead(yarp::os::Bo
ttle& b)
{
    if (b.size() == 0)
        return;

    sens->gripperMutex.lock();
    if(b.get(14).asDouble() > THRESHOLD_GRIPPER &&
b.get(14).asDouble() < 1000)
        {sens->gripper=true;
         CD_INFO("Botella detectada\n");}
    else
       sens->gripper=false;
    sens->gripperMutex.unlock();

      double max=0;
      for(int i=0;i<b.size();i++)
      {if(b.get(i).asDouble()>max)
      max=b.get(i).asDouble();}
        CD_INFO("Current maximum sensor value: %f\n", max);
    sens->alertMutex.lock();
    if(max > THRESHOLD_ALERT)
    sens->alert = HIGH;
        else if (max > THRESHOLD_LOW_ALERT)
        {sens->alert = LOW;}
            else
            sens->alert = ZERO;

//    sens->alert = max > THRESHOLD_ALERT;
    sens->alertMutex.unlock();
}
```

Anexo 11. ProximitySensorsClient.hpp

```
// -*- mode:C++; tab-width:4; c-basic-offset:4; indent-tabs-mode:nil -
*-

#ifndef __PROXIMITY_SENSORS_HPP__
#define __PROXIMITY_SENSORS_HPP__

#include <yarp/os/all.h>
#include <yarp/dev/Drivers.h>
#include <yarp/dev/PolyDriver.h>

#include "IProximitySensors.h"

#define DEFAULT_LOCAL "/sensor_reader"
#define DEFAULT_REMOTE "/serial/out"

#define DEFAULT_PORTMONITOR_TYPE "lua"
#define DEFAULT_PORTMONITOR_CONTEXT "sensors"
#define DEFAULT_PORTMONITOR_FILE "amor_sensors_modifier"

namespace roboticslab
{

/**
 * @ingroup TeoYarp
 * \defgroup ProximitySensors
 *
 * @brief Contains roboticslab::ProximitySensors.
 */

/**
 * @ingroup ProximitySensors
 * @brief The ProximitySensors class implements IProximitySensors.
 */
class ProximitySensorsClient : public yarp::dev::DeviceDriver, public
IProximitySensors
{
    public:

        ProximitySensorsClient() : alert(ZERO), gripper(false)
        {}

        // -------- IProximitySensors declarations.
            // --------Implementation in IProximitySensorsImpl.cpp ----
----

        virtual alert_level getAlertLevel();
        virtual bool hasTarget();

        // -------- DeviceDriver declarations.
            // -------- Implementation in IDeviceImpl.cpp --------

        virtual bool open(yarp::os::Searchable& config);

        /**
        * Close the DeviceDriver.
        * @return true/false on success/failure.
        */
        virtual bool close();

        static const int THRESHOLD_GRIPPER;
        static const int THRESHOLD_ALERT;
```

```
        static const int THRESHOLD_LOW_ALERT;

    protected:

        class SensorReader : public
yarp::os::BufferedPort<yarp::os::Bottle>
        {
        public:
            SensorReader() : sens(NULL) {}
            virtual void onRead(yarp::os::Bottle& b);
            void setReference(ProximitySensorsClient * sens)
            {
                this->sens = sens;
            }
        private:
            ProximitySensorsClient * sens;
        };

        SensorReader sr;
        alert_level alert;
        bool gripper;
        yarp::os::Mutex alertMutex, gripperMutex;

};

}  // namespace roboticslab

#endif  // __PROXIMITY_SENSORS_HPP__
```

Anexo 12. IProximitySensorsImpl.cpp

```
// -*- mode:C++; tab-width:4; c-basic-offset:4; indent-tabs-mode:nil -
*-

#include "ProximitySensorsClient.hpp"

#include <ColorDebug.hpp>

roboticslab::IProximitySensors::alert_level
roboticslab::ProximitySensorsClient::getAlertLevel()
{
    alertMutex.lock();
    alert_level alert_copy = alert;
    alertMutex.unlock();
    return alert_copy;
}

bool roboticslab::ProximitySensorsClient::hasTarget()
{
    gripperMutex.lock();
    bool gripper_copy = gripper;
    gripperMutex.unlock();
    return gripper_copy;
}
```

Anexo 13. Guía de instalación de los módulos

En este anexo se incluyen las indicaciones, paso a paso, para realizar la instalación y ejecución de los módulos que componen nuestro programa.

En primer lugar, debemos asegurarnos que todos los elementos físicos del sistema están correctamente conectados:

- Comprobación del conexionado de cables para el uso del robot: CAN, SpaceNavigator, sensores a las placas Arduino…

- Configuración de la fuente de alimentación de los sensores a 12 V.

- Conexionado del robot a su fuente de alimentación propia. Comprobación de la seta de seguridad del robot.

A continuación, comenzaremos la instalación y ejecución de los programas desarrollados en Linux:

- Conexionado para el control del robot por joystick, a través del bus CAN. En una terminal vacía, independiente de la ubicación de la carpeta con el programa, escribir el comando indicado a continuación. Este comando se lanzará una única vez por sesión.

```
sudo ip link set can0 up txqueuelen 1000 type can bitrate 1000000
```

- Comprobación del funcionamiento del robot. Se ejecutará el programa *exampleAmor*, proporcionado por el fabricante, para probar los movimientos del robot.

  IMPORTANTE: Debemos desconectar la seta de seguridad para permitir los movimientos del robot. Además, nos aseguraremos de que el robot se encuentre sobre una superficie blanda, para evitar que, en caso de fallo, este se precipite sobre una superficie que pueda causarle daños.

  En caso de error, debemos comprobar que los frenos de las articulaciones del robot están desactivados. En caso de que haya que hacer una desconexión manual, ejecutar el siguiente comando:

  Freno de la articulación 1:

  ```
  cansend can0 033#00
  ```

  Freno de la articulación 2:

  ```
  cansend can0 032#00
  ```

- Realizar la conexión con YARP. En una nueva terminal, independiente de la ubicación del programa, indicar el siguiente comando:

```
yarpserver --write
```

- Enlazar sensores. En una nueva terminal, independiente de la ubicación del programa, indicar el siguiente comando:

```
sudo yarpdev --device serialport --comport /dev/ttyACM0 --baudrate 115200 --paritymode NONE --readtimeoutmsec 3000 --xinenb 0 --xoutenb 0 --databits 8 --stopbits 1
```

- Abrir la GUI con la interfaz gráfica que muestra los datos se los sensores en una tabla organizada por zona del brazo y número de sensor. En una nueva terminal, modificaremos nuestra ubicación al escritorio con el comando *cd Desktop*, y escribiremos:

```
python sensor_reader.py
```

- Realizar la conexión entre los datos recibidos por los puertos habilitados para la comunicación y nuestra interfaz gráfica (GUI), así como con el procesado de los datos. En una terminal disponible, modificaremos nuestra ubicación al escritorio con el comando *cd Desktop*, y escribiremos:

```
yarp connect /serial/out /sensor_reader_gui tcp+recv.portmonitor+type.lua+file.amor_sensors_modifier.lua
```

- Sacar a la vista la gráfica X/Y con el comando *yarpscope.* En una nueva terminal, modificaremos nuestra ubicación al escritorio con el comando *cd Desktop*, y escribiremos:

```
yarpscope --index "(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47)" --min 0 --max 1000
```

- Realizar la conexión entre los datos recibidos por los puertos habilitados para la comunicación y el yarpscope, así como con el procesado de los datos. En una terminal disponible, modificaremos nuestra ubicación al escritorio con el comando *cd Desktop*, y escribiremos:

```
yarp connect /serial/out /yarpscope tcp+recv.portmonitor+type.lua+file.amor_sensors_modifier.lua
```

Una vez finalizados estos pasos, ya tenemos conectado el sistema para su utilización. A continuación, comenzaremos con la instalación de los programas necesarios para nuestro proyecto.

- Abrir el plug-in de control del robot. En una terminal disponible, independiente de la ubicación del programa, indicar el siguiente comando:

```
yarpdev --device AmorCartesianControl
```

- Abrir el plugin del space navigator. En una terminal disponible, independiente de la ubicación del programa, indicar el siguiente comando:

```
yarpdev --device SpaceNavigator --period 5 --name /spacenavigator --
ports "(mouse buttons)" --channels 8 --mouse 0 5 0 5 --buttons 6 7 0 1
```

- Abrir programa de control conjunto (SpaceNavigator + robot + sensores). En una terminal disponible, modificaremos nuestra ubicación al escritorio con el comando *cd Desktop*, y escribiremos:

```
streamingSpnav --remoteCartesian /CartesianControl --scaling 10
```

- Realizar la conexión entre este programa y los sensores, mediante los puertos habilitados para la comunicación.

```
yarp connect /serial/out /sensor_reader
tcp+recv.portmonitor+type.lua+file.amor_sensors_modifier.lua
```

NOTA: Los puntos 8, 10 y 14 podrían evitar tener que abrirse todo el rato para diferentes pruebas, utilizando una conexión persistente añadiendo en una terminal el siguiente comando:

```
yarp connect --persist /serial/out /sensor_reader
tcp+recv.portmonitor+type.lua+context.sensors+file.amor_sensors_modifi
er.lua
```

Anexo 14. Especificaciones técnicas robot AMOR

GENERAL	
Grados de Libertad	7
Carga máxima	2,5 kg
Alcance	95 cm
Efector final	Garra con dos dedos (Opcional)
Feedback de la posición del motor	Resolución de al menos 0,16º
Encoder rotacional absoluto	Resolución de 0,3º
SEGURIDAD	
Embrague deslizante	En las articulaciones de la base y hombro
Seguridad intrínseca	Garra con apertura/cierre controlables
DIMENSIONES Y PESO	
Peso	9 kg
Tamaño	120x85x20 mm
Transporte	Caja de transporte de carcasa dura rellena de espuma
INTERFAZ	
Tipo	1 Mbit/s CAN
Conector	D-SUB 9 hembra
Funcionalidad	Control de voltaje PWM del motor de bajo nivel y alta velocidad (superior a 1kHz). PID de control de posición.
Feedback de la posición	Feedback superior a 1kHz
Consumo	24 V DC, 5A
FUNCIONALIDAD DE LA API	
Sistema Operativo	Microsoft Windows
Control de posición articular	SI
Control de velocidad articular	SI
Control de posición cartesiana	SI
Control de velocidad cartesiana	SI
AMBIENTE Y ALREDEDORES	
Ambiente de trabajo	Evita suciedad, restos de aceite y gases corrosivos
Temperatura de trabajo	0-50º C
Humedad	< 80% TH, sin condensación, sin escarcha
Vibración	3G máximo
Temperatura de almacenamiento	de -20º a 70º C